UNIVERSITETET I OSLO
Institutt for informatikk

Condor

- et rammeverk for mobilitetsbaserte kontekstoppmerksomme applikasjoner

Richard Moe Gustavsen

Hovedfagsoppgave

1. august 2002

Forord

Denne hovedfagsoppgaven har pågått over fire semestre, fra august 2000 til august 2002, og er skrevet som en del av graden Candidatus Scientiarum ved Institutt for informatikk ved Universitetet i Oslo. Jeg vil først og fremst benytte denne siden til å rette en stor takk til Arne-Jørgen Berre som har vært min hovedveileder gjennom oppgaven. En spesiell takk går også til Odd-Wiking Rahlff som har bidratt med informativ veiledningen fra uke til uke, og vist smittende entusiasme og engasjement underveis.

Jeg vil også benytte anledningen til å takke en rekke andre personer som på ett eller annet vis har vært involvert. Takk til Athletic Gateway v/Bjørn og Stein Erling Birkeland for interessen rundt Mobitras. Takk til Astrid Nordhagen og Jan Håvard Skjetne for uttestingen av Mobitras rundt Sognsvann. Takk til Erik Nilsson, Børge Haugset, Fredrik Vraalsen, og ellers andre hos SINTEF som fra tid til annen har deltatt på mine veiledningsmøter.

Til sist vil jeg også takke Arild Endresen og Hilde Bjelland Nilsen for å ha lest igjennom oppgaven usedvanlig nøye helt på tampen.

Oslo, 29.07.2002

Sammendrag

I denne oppgaven har jeg studert mobilitetsbaserte, kontekstoppmerksomme applikasjoner. Kort fortalt er dette applikasjoner som er utviklet for å støtte mobile PDA-brukere med tjenester basert på hvem de er, hvor de er, og hva de holder på med. Jeg har også valgt å rette oppmerksomheten mot én slik tjeneste, nemlig anvendelse av kontekstrelevant informasjon. Dette er informasjon som er spesielt relevant for PDA-brukeren på grunn av situasjonen han befinner seg i.

Målet med oppgaven var å konstruere et overordnet rammeverk for applikasjonstypene jeg rettet søkelyset mot. For å tilnærme meg dette, analyserte jeg en rekke eksempelapplikasjoner. Jeg så på hva slags kontekstuell informasjon som var relevant for disse, hvordan denne informasjonen kunne registreres, hva den ble anvendt til, og hvordan den kunne benyttes sammen med bruk av kontekstrelevant informasjon. Resultat av analysen var en kravspesifikasjon for et rammeverk.

Med utgangspunkt i kravspesifikasjonen ovenfor, konstruerte jeg et overordnet rammeverk ved navn Condor. En viktig observasjon bak dette løsningsforslaget, var at kontekstuell-, og kontekstrelevant informasjon ble anvendt meget likt blant eksempelapplikasjonene. Det som i ett tilfelle ble sett på som kontekstuell informasjon (som for eksempel PDA-brukers posisjon), kunne i neste betraktes som kontekstrelevant informasjon. Jeg vurderte av den grunn denne informasjonen som en og samme abstraksjon. I Condor demonstrerte jeg hvordan denne abstraksjonen kunne realiseres ved hjelp av det jeg benevnte som kontekstdokumenter. Sammen med en utfyllende arkitektur vurderte jeg Condor til å tilfredsstille de fleste av kravene nedfelt i kravspesifikasjonen.

Innhold

Figurer

Kapittel 1

Innledning

I løpet av de siste årene har bruken av håndholdte datamaskiner blitt mer og mer vanlig. Slike maskiner er ofte små slik at de kan oppbevares i jakkelommen, men likevel så kraftige at de kan eksekvere relativt avanserte dataprogrammer. Da denne oppgaven gikk i trykken veide den nyeste versjonen av Cassiopeia Pocket-Pc[1] 190 gram, hadde en 206Mhz prosessor, og kunne oppbevare inntil 64MB data (se Figur 1). Slike håndholdte datamaskiner har fått tildelt fellesbetegnelsen PDA-er (*Personal Digital Assistants*).

Figur 1: Cassiopeia E-200 – Et eksempel på en PDA

En vesentlig forskjell som skiller PDA-ene fra andre typer bærbare maskiner, er at man enklere kan ta dem med og benytte dem *mens* man beveger seg rundt. Man trenger ikke nødvendigvis å stoppe opp eller sette seg ned på forhånd [Thanh 2001]. Dette gjør at man også kan forestille seg en rekke nye situasjoner hvor det kan være relevant å anvende datakraft. Med Cassiopeiaen er det for eksempel en enkel prosess å legge inn nye avtaler i den innebygde kalenderen, selv om man skulle befinne seg på en spasertur utendørs.

---

[1] Informasjon om denne kan leses hos "*http://www.casio.com/personalpcs/*" (dato: 29.07.2002).

I kjølvannet av PDA-ene har det vokst frem en interesse for en ny type applikasjoner. Kort forklart er de nye applikasjonene konstruert for være oppmerksomme på PDA-brukerens omgivelser, noe som inkluderer blant annet hans posisjon. Ved hjelp av slike opplysninger kan de produsere en rekke *kontekstrelevante tjenester* tilbake. Dette er tjenester som er spesielt relevante for brukeren på grunn av situasjonen han befinner seg i [Dey 2000]. En slik tjeneste kan for eksempel inkludere opptegning av hans posisjon på et kart, eller presentasjon av en serverdighet i hans omgivelser. For å gi PDA-ene tilgang på kontekstuell informasjon, kobles de ofte til sensorer. Dette gjøres fordi PDA-ene vanligvis ikke har noen egen innebygd funksjonalitet for å registrere brukerens[1] situasjon. En utbredt metode for å bestemme posisjon, for eksempel, er å anvende en separat GPS-mottaker[2].

Dette området synes jeg er spennende: Applikasjoner som er oppmerksomme på brukerens situasjon. Jeg har derfor valgt å la oppgaven dreie seg om nettopp slike applikasjoner. Jeg ønsker å finne ut av hva som særpreger dem, hvilken funksjonalitet de vanligvis anvender, og hvordan utvikling av dem kan støttes ved hjelp av et *rammeverk*. Nøyaktig hva et rammeverk er vil jeg komme tilbake til senere, men kort forklart kan det oppsummeres som en gjenbrukbar komponentmodell som tilbyr en abstrakt design for å løse et sett med relaterte problemer [Bosch et al. 1999]. En slags halvveis utviklet applikasjon, med andre ord.

Jeg vil fortsette dette avsnittet med et eksempel. Eksempelet skal gi leseren et mer konkret bilde på hvilke type applikasjoner jeg retter søkelyset mot.

> *”Kari har tatt seg fri fra jobben og er på ferie i Oslo. Etter en frokost på hotellet har hun nå kommet frem til dagens hovedmål; Vigelandsparken på Frogner. Med seg i jakkelommen har hun sin private PDA, og på skulderen har hun en tilkoblet GPS-mottaker. PDA-en er for anledningen forhåndsinstallert med turistinformasjon for de fleste av attraksjonene i byen, inkludert statuene i Vigelandsparken.*
>
> *På grunn av mottakeren har applikasjonen god kontroll på hvor Kari befinner seg. En av dens hovedoppgaver er derfor å tegne hennes posisjon opp på et kart. Dermed ser hun bestandig hvor hun er, hvor hun har vært, og hvor hun skal.*

---

[1] Med ”bruker” mener jeg alltid en ”PDA-bruker” om ikke annet kommer klart frem i teksten.

[2] GPS (*Global Positioning System*) er en gjennomgående metode blant oppgavens applikasjoner for å bestemme lokasjon utendørs (bare utendørs, ikke innendørs). Kort fortalt er det et system bestående av tjueseks satellitter som kretser i ulike baner rundt jorden, og som sender ut tidsstemplede signaler som tilhørende mottakere verden over kan fange opp. Disse anvendes ved hjelp av triangulering hos hver enkelt mottaker til å bestemme deres egen lokasjon. Mer informasjon om GPS kan leses hos http://www.nasa.gov/ (dato: 29.07.2002).

Kari har akkurat fått øye på Sinnataggen. På vei bort blir informasjon om skulpturen hentet frem på PDA-en, mens en lyd spilles av for å gjøre henne oppmerksom på meldingen. Opplysningene som dukker frem består av et bilde av skulpturen tatt av en profesjonell fotograf, og en knapp som ved aktivering spiller av et innlest opptak om skulpturens bakgrunn.

Applikasjonen kan mer enn å bare presentere informasjon i henhold til Karis posisjon. Den gir henne også muligheten til å forfatte sine egne notater. Bildene hun tar med PDA-ens innebygde digitalkamera, vil for eksempel legge seg som små ikoner på kartet der de ble tatt. Ved å fysisk gå bort til disse, eller trykke på skjermen der de ligger, vil de dukke frem igjen. De oppfører seg med andre ord nøyaktig likt som de andre meldingene. Slike notater bevarer gjerne minner og historie over hvor man har vært, noe Kari liker å hente frem igjen ved senere anledninger."

Eksempelet over er hentet fra Mocado. Dette er en av to applikasjoner jeg har valgt å utvikle som en del av oppgaven. Eksempelet viser spesielt hvordan kunnskap om posisjon kan skape et utgangspunkt for en ny type applikasjoner. Disse benevnes som *mobilitetsbaserte*, og kjennetegnes ved at en eller flere av tjenestene de tilbyr er basert på brukerens mulighet til forflytte seg i det fysiske rom [Thanh 2001]. Mocado er en slik applikasjon. Både det å tegne opp Karis posisjon på et kart, samt vise frem informasjon om en statue hun observerer, kan vurderes som selvstendige mobilitetsbaserte tjenester.

Selv om lokasjon[1] betraktes som en særskilt viktig parameter blant mobilitetsbaserte applikasjoner [Dey et al 1998, Salber et al. 1998, Brown 1995], er den likevel ikke alene. Når man endrer lokasjon, endres gjerne situasjon på andre måter. Nye gjenstander i omgivelsene kommer til, andre forsvinner. Forskjellige sosiale situasjoner oppstår, og man kommuniserer med nye personer. I tillegg kan informasjon om temperatur, hjerterytme, lyd-, og lysforhold og lignende, også være interessant. Nedenfor vil jeg benevne alle slike opplysninger som sammen beskriver en brukers situasjon, for hans *kontekst*. Mer generelt vil jeg derfor

[1] Gjennomgående i oppgaven vil jeg anvende begrepene posisjon og lokasjon. I de tilfellene der jeg sikter til et bestemt koordinatpar, vil jeg anvende "posisjon". I andre tilfeller, hvor en posisjon isteden kan bestå av et areal, vil jeg anvende "lokasjon".

konsentrere meg om applikasjoner som er oppmerksomme på brukerens kontekst, fremfor bare hans posisjon. Av den grunn vil jeg også relativt raskt komme inn på *kontekstoppmerksomhet*[1]. Grovt skissert omhandler dette området applikasjoner som er oppmerksomme på brukerens kontekst for å tilby kontekstrelevante tjenester [Dey 2000]. Jeg vil videre rette søkelyset på kun én slik tjenestekategori: Anvendelse av *kontekstrelevant informasjon*. Kontekstrelevant informasjon er informasjon som er spesielt relevant for brukeren når han befinner seg i en forhåndsbestemt situasjon. Slik informasjon er med andre ord assosiert til en kontekst [Dey 2000]. Notatene som ble anvendt i eksempelet med Kari, var slik informasjon. Når hun befant seg i nærheten av et sted der et notat lå plassert, skulle notatet fremvises, og bare da.

Oppsummert vil jeg studere mobilitetsbaserte kontekstoppmerksomme applikasjoner som tilbyr kontekstrelevant informasjon. Begrepene ”kontekst”, ”kontekstoppmerksomhet”, og ”kontekstrelevant informasjon” er derfor meget sentrale i oppgaven, og vil av den grunn også bli drøftet utførlig nedenfor.

1.1 Behov for rammeverk

Behovet for å anvende kontekstoppmerksomhet viser seg å være relevant for flere ulike applikasjonsområder [Lieberman et al. 2000]. Likevel utnyttes det relativt sjeldent [Dey 2000, Pascoe 1998]. Dette inkluderer også mobilitetsbaserte applikasjoner. I den grad kontekstoppmerksomhet anvendes, dreier det seg stort sett bare om lokasjon [Pascoe 1998, Cheverst et al. 2000, Abowd et al. 1996]. En av årsakene til dette forklares ofte med vanskeligheten det er å benytte sensorer og kontekst generelt [Dey 2000, Pascoe 1998, Salber et al.1998, Hong 2000]. I de tilfellene hvor slik oppmerksomhet har blitt realisert, har implementasjonen ofte blitt tilfeldig: Fremfor at en generell struktur for anvendelse av kontekstoppmerksomhet er blitt konstruert, har man heller kommet frem til en løsning der skillet mellom sensorene og resten av applikasjonen er blitt lav [Dey 2000]. Dette hindrer gjerne gjenbruk av design og kildekode ved utvikling av nye kontekstoppmerksomme applikasjoner.

Med det ovenstående som utgangspunkt, vil jeg altså fordype meg i mobilitetsbaserte kontekstoppmerksomme applikasjoner. Jeg vil studere både hva slags funksjonalitet de tilbyr, med også hva man har behov for å la de tilby. Jeg vil fokusere på bruk av kontekstrelevant informasjon, og samhandlingen mellom dette og kontekstoppmerksomhet. Målet er å konstruere et overordnet rammeverk for utvikling av slike applikasjoner. Jeg vil med andre ord *ikke* forsøke å implementere og teste ut et rammeverk, men isteden komme frem til en overordnet og initsiell konstruksjon.

[1] Den engelskspråklige betegnelsen på området er *context-aware applications*.

1.2 Problemstilling

Jeg vil belyse følgende problemstilling med oppgaven:

> *Jeg vil studere mobilitetsbaserte kontekstoppmerksomme applikasjoner som behandler kontekstrelevant informasjon, og hvordan et rammeverk for slike kan utformes.*

Begrepsavklaring

Med mobilitetsbaserte applikasjoner mener jeg applikasjoner som er spesielt konstruert for å tilby tjenester basert på at brukeren er mobil. Jeg vil drøfte nærmere hva dette innebærer i Kapittel 3.4. Med kontekstoppmerksomme applikasjoner mener jeg applikasjoner som er oppmerksomme på brukerens kontekstuelle situasjon. Jeg vil drøfte både kontekst og kontekstoppmerksomhet mer utfyllende i Kapittel 3. Med kontekstrelevant informasjon mener jeg informasjon som er relevant for brukeren når han befinner seg i en gitt kontekstuell situasjon. Dette begrepet vil bli drøftet i Kapittel 3.3. Med rammeverk mener jeg en gjenbrukbar komponentmodell. En slik modell består av et sett med klasser som tilbyr en abstrakt design for å løse et sett med relaterte problemer [Bosch et al. 1999]. Jeg vil belyse rammeverk mer omfattende og generelt i Kapittel 2.

Delproblemer

Det er i hovedsak tre delproblemer jeg mener må besvares før jeg kan gi en konklusjon på problemstillingen. Disse vil jeg for ordens skyld behandle sekvensielt:

Hvordan anvendes kontekstoppmerksomhet? For å utvikle et rammeverk som anvender kontekstoppmerksomhet, må jeg tilegne meg gode kunnskaper på området. Jeg må vite hva kontekstoppmerksomhet er, hva slags kontekstuell informasjon som er interessant, og hvordan dette kan registreres

Hvordan anvendes kontekstrelevant informasjon? Slik som for kravet ovenfor må jeg også gjøre meg kjent meg bruk av kontekstrelevant informasjon. Jeg må vite hva slags egenskaper slik informasjon har, hvordan den ønskes anvendt, og hvordan denne anvendelsen kan integreres med kontekstoppmerksomhet.

Hvordan kan et rammeverk konstrueres slik at den identifiserte funksjonaliteten støttes? Etter å ha behandlet delspørsmål en og to, vil jeg komme frem til en kravspesifikasjon. Denne spesifiserer hva slags funksjonalitet et rammeverk for applikasjonene jeg har rettet fokus på, bør tilfredsstille. Spørsmålet som vil gjøre seg gjeldene her, er hvordan et slikt rammeverk kan realiseres. Jeg vil undersøke hvilke abstraksjoner som er sentrale, hvilke komponenter som bør være med, og hvordan disse bør samhandle. For å få til dette, vil jeg blant annet undersøke hvordan andre lignende rammeverk har støttet slike applikasjoner tidligere. Svarene vil danne utgangspunktet for mitt eget løsningsforslag, ”Condor”.

1.3 Kapittelinndeling

Kapittel 1: Innledning – Dette kapitlet.

Kapittel 2: Litt om rammeverk – Denne oppgaven skal dreie seg om utviklingen av et konkret rammeverk. I Kapittel to vil jeg drøfte hva et rammeverk er, noen sentrale begreper på området, hvilke problemer man kan møte under utviklingen av et rammeverk, noen ulike utviklingsmodeller, og hvilken metode jeg har valgt å følge i denne oppgaven.

Kapittel 3: Kontekst og kontekstoppmerksomhet – Dette er to sentrale begreper i oppgaven. Jeg vil derfor benytte kapittel tre til å belyse disse. Til sist vil jeg også se på anvendelse av kontekst og kontekstoppmerksomhet blant mobilitetsbaserte applikasjoner. Svarene fra dette kapitlet vil danne utgangspunktet for hvilke eksempelapplikasjoner jeg vil studere og utvikle.

Kapittel 4: Fire eksterne eksempelapplikasjoner – Etter å ha gjort meg kjent med oppgavens område, er turen kommet for å studere eksterne eksempelapplikasjoner. I kapittel fire vil jeg gi en kort presentasjon av de fire jeg har valgt ut.

Kapittel 5: Mocado – I løpet av studiet har jeg utviklet to selvstendige eksempelapplikasjoner. I kapitel fem vil jeg presentere den første, Mocado. Jeg vil drøfte all relevant funksjonalitet hos denne, både det som har fungert tilfredsstillende, men også problemer, mangler, ønsker, og feil.

Kapittel 6: Mobitras – Dette er den andre applikasjonen jeg har utviklet. Mens Mocado er en slags turistapplikasjon[1], er Mobitras isteden tiltenkt idrettsutøvere og mosjonister. I kapitel seks vil jeg vil presentere all relevant funksjonalitet hos denne applikasjonen, og drøfte hva som har fungert tilfredsstillende, samt problemer, mangler, ønsker, og feil.

Kapittel 7: Behandling av kontekstoppmerksomhet – Etter å drøftet sentrale begreper i oppgaven, samt presentert dens eksempelapplikasjoner, er jeg klar for å analysere anvendelse av kontekstoppmerksomhet. Dette inkluderer blant annet hvordan kontekst kan registreres. Resultatene herfra vil danne utgangspunktet for første halvdel av kravspesifikasjonen i Kapittel 9.

Kapittel 8: Behandling av kontekstrelevant informasjon – I kapitel åtte vil drøfte anvendelse av kontekstrelevant informasjon, og analysere og vurdere hvilke egenskaper slik informasjon bør ha. Resultatene herfra vil danne utgangspunktet for resten kravspesifikasjonen i Kapittel 9.

Kapittel 9: Kravspesifikasjon – I kapitel ni presenterer jeg en kravspesifikasjon som oppsummerer og konkretiserer resultatene fra kapitel syv og åtte.

Kapittel 10: Tre eksterne rammeverk – Etter å ha utformet en kravspesifikasjon, er jeg interessert i å undersøke om det eksisterer noen rammeverk som støtter denne. Hvis det er det, vil dette svekke grunnen for å utvikle et nytt. Jeg vil også

[1] Med ”turistapplikasjon” mener jeg en applikasjon som fremviser informasjon om attraksjoner i nærheten av brukeren. Jeg vil benytte denne betegnelsen flere steder i oppgaven.

studere hvordan løsningene jeg velger ut er konstruert. Dette gjøres med tanke på gjenbruk av design. Kapitlet er derfor todelt; jeg vil både undersøke hva andre har gjort før meg, men også foreta en designanalyse.

Kapittel 11: Løsningsforslag: Condor – I kapittel elleve vil jeg presentere mitt eget løsningsforslag, Condor. Jeg vil drøfte noen gjennomgående observasjoner jeg mener å identifisert i løpet av oppgaven, og videre presentere et rammeverk basert på disse. Til sist vil jeg drøfte løsningen opp mot kravspesifikasjonen for å undersøke i hvilken grad kravene ble oppfylt.

Kapittel 12: Konklusjon og veien videre – Avslutningsvis vil drøfte konklusjonen jeg har kommet frem til på problemstillingen, og peke på noen mulige retninger for videre arbeid.

Kapittel 2

Litt om rammeverk

Denne oppgaven skal dreie seg om utviklingen av et rammeverk. Jeg vil av den grunn benytte dette kapitlet til å drøfte noen sentrale begreper rundt dette området. Jeg vil også vurdere noen ulike metoder for utvikling av rammeverk, samt belyse hvordan jeg har valgt å tilnærme meg Condor.

2.1 Definisjoner og innhold

Fayad hevder at det ikke finnes noen allment akseptert definisjon på hva et rammeverk er [Fayad et al. 1999]. Likevel er det bred enighet om at formålet med dem først og fremst er å oppnå gjenbruk av design og kildekode [Fayad et al. 1999, Bosch et al. 1999]. Fremfor å konstruere nye applikasjoner innen en bestemt kategori fra bunnen av hver gang, kan man forsøke å identifisere hva som er generelt og felles hos disse. Dette kan sammenfattes og konstrueres som en slags halvveis utviklet applikasjon, benevnt rammeverk. Når man senere ønsker å ferdigstille en mer konkret applikasjon, kan man heller utvide og spesialisere rammeverket. En slik spesialisering vil jeg i denne oppgaven benevne som en *klientapplikasjon*. Den er en klient i forhold til rammeverket den anvender. Ved å utvikle klientapplikasjoner kan man oppnå kortere produksjonstid, lavere produksjonskostnader, og bedre programvarekvalitet [Fayad et al. 1999]. Et godt utviklet rammeverk kan redusere mengden kildekode hos en klientapplikasjon med opptil nitti prosent [Fontoura et al. 2002]. Et eksempel på et rammeverk er AWT (”Abstract Window Toolkit”) for Java fra Sun[1]. Dette kan benyttes til å konstruere grafiske brukergrensesnitt. Her finner man ferdige komponenter som vinduer, menyer, knapper, radioknapper, og lignende. Disse kan

[1] Mer om AWT kan leses hos http://java.sun.com/j2se/1.3/docs/guide/awt/ (dato: 29.07.2002).

settes sammen slik man vil at grensesnittet skal se ut, fremfor å bygge alt fra bunnen av selv.

Som nevnt finnes det flere ulike definisjoner på rammeverk. Roberts anvender følgende:

> *"Frameworks are reusable designs of all or part of a software system descibed by a set of abstract classes and the way instances of those classes collaborate"*
>
> – [Roberts et al. 1996, side 1]

Denne definisjonen ekskluderer imidlertid det som benevnes som et *komponentbibliotek* som en del av rammeverket. Et komponentbibliotek består av ferdigkonstruerte komponenter som er utviklet ved hjelp av rammeverket [Bosch et al. 1999]. I AWT er dette typisk knapper, menyer, og lignende. Dette er komponenter man ikke trenger å spesialisere for å anvende. Tilpasning til hver enkelt applikasjon skjer isteden ved hjelp av metodekall med argumenter. Når et rammeverk kan anvendes på denne måten, uten å måtte spesialisere noen av dens klasser, benevnes de som *black box* [Pree 1999, Bosch et al. 1999, Roberts et al. 1996]. Dette vil si, applikasjonsutviklerne trenger ikke noen dypere forståelse av hvordan rammeverket er konstruert for å kunne anvende det. Det motsatte, *white box*, vil isteden bety et rammeverk der utviklerne må spesialisere dets abstrakte klasser ved anvendelse [Pree 1999, Bosch et al. 1999, Roberts et al. 1996]. Som oftest vil de første versjonene av et rammeverk være av denne typen. Etter hvert som det bygges ut med flere og flere komponenter i et komponentbibliotek vil det utvikles til å bli mer i retning av black box [Fayad et al. 1999]. I likhet med Bosch vurderer også jeg rammeverkskomponentene som en del av rammeverket. Disse to delene mener jeg hører naturlig sammen. Av den grunn vil jeg også anvende hans definisjon på et rammeverk:

> *"A framework is a set of classes that embodies an abstract design for soulutions to a family of related problems"*
>
> – [Bosch et al. 1999, side 58]

Bosch skiller mellom det han benevner som *kjernerammeverket*, og dens *inkrementer* [Bosch et al. 1999]. Det første er hovedrammeverket uten de ferdige komponentene, mens det siste er de omtalte ferdigimplementerte komponentene som legges til etter hvert. Denne definisjonen inkluderer også konkrete klasser som en del av kjernerammeverket, og ikke bare abstrakte versjoner slik som tilfellet var i den første definisjonen.

En egen måte å kategorisere rammeverk på er etter i hvilken grad de er horisontale eller vertikale [Rogers 1997]. Et vertikalt rammeverk er et rammeverk som bare er anvendelig for en smal kategori med applikasjoner, mens et horisontalt

kan anvendes mer på tvers av slike. Java AWT var et eksempel på et horisontalt rammeverk. Enhver Java-applikasjon som skal benytte et grafisk brukergrensesnitt, kan anvende det. I denne oppgaven vil jeg vurdere Condor som mer vertikalt enn horisontalt. Dette er fordi det i utgangpunktet er begrenset til å bare støtte utviklingen av en relativt smal gruppe med applikasjoner, og da først og fremst de som skal presentere kontekstrelevant informasjon. Imidlertid kan kontekstoppmerksomhet[1] isolert sett anvendes blant flere applikasjonstyper, og til mange ulike formål [Lieberman et al. 2000]. Dette gjør Condor også horisontalt.

En viktig karakteristikk ved et rammeverk er hvordan kontrollflyten forflytter seg. I tradisjonell applikasjonsutvikling (uten rammeverk) er det applikasjonsutvikleren som har ansvaret for dette. Han kjenner hele applikasjonsstrukturen og bestemmer hvordan komponentene skal samhandle seg imellom. For rammeverk kan dette være annerledes. Dette kommer av at utvikleren kun har til oppgave å spesialisere rammeverkets abstrakte klasser, eller sette sammen dets inkrementer, uten å ta hensyn til hvordan kontrollen vil forflytte seg under eksekvering. De spesialiserte klassene blir isteden aktivert ved hjelp av hendelser (*events*) som gjør at rammeverket foretar kall på de abstrakte eller virtuelle metodene hos spesialiseringene. Det er med andre ord rammeverket som får ansvaret for kontrollflyten under eksekvering.

2.2 Utvikling av rammeverk

> *"People develop abstractions by generalizing from concrete examples. Every attempt to determine the correct abstractions on paper without actually developing a running system is doomed to failure. No one is that smart. A framework is a reusable design, so you develop it by looking at the things it is supposed to be a design of. The more examples you look at, the more general your framework will be."*
>
> – [Roberts 1996, side 2]

Utvikling av rammeverk vurderes normalt som en vanskelig prosess [Aksit et al. 1999, Roberts et al. 1996]. Dette har flere årsaker. Først og fremst må et rammeverk fange generelle krav på tvers av flere typer applikasjoner innen et gitt område. Ikke bare ett, slik som er tilfellet ved vanlig applikasjonsutvikling (noe som kan være vanskelig nok i seg selv [Sommerville 1998]). I tillegg består rammeverk hovedsakelig av abstraksjoner [Fayad et al. 1999]. Abstraksjonene representerer det som er felles og generelt. På grunn av dette kan de være vanske-

[1] Både kontekstoppmerksomhet og kontekstrelevant informasjon vil jeg definere og drøfte i neste Kapittel.

lige å verifisere; de beskriver nemlig ikke noe konkret som de kan vurderes mot. Om et rammeverk er ferdigutviklet lar seg derfor vanskelig avgjøre uten å ha testet det ut på noen reelle applikasjoner [Fayad et al. 1999]. Likevel er det uheldig å benytte et rammeverk før det er helt ferdig [Fayad et al. 1999]. Endrer man på rammeverket etter dette, kan dette medføre at også klientapplikasjonene endres. På den andre siden er det en vanlig oppfatning at man må begynne å anvende rammeverket om det skal fortsette å utvikle seg [Fayad et al. 1999, Pree 1999, Roberts et al. 1996]. Erfaringene man får under denne prosessen er svært viktig, og resulterer ofte i at løsningen blir mer i retning av black box [Roberts et al. 1996]. En av de vanligste observasjonene som derfor gjøres, er at et rammeverk krever mange iterasjoner før det vurderes som ferdig. Dette er også en vanlig fase under tradisjonell programvareutvikling, men utgjør likevel en mer markant del av utviklingsprosessen for rammeverk [Fayad et al. 1999].

I begynnelsen av utviklingen begynner man vanligvis alltid med en analyse av rammeverksområdet[1] [Fayad et al. 1999]. Dette er en fellesbetegnelse på all informasjon som beskriver eller forklarer applikasjonene rammeverket skal støtte. Denne informasjonen innhentes hovedsakelig fra to kilder: problemområdet og løsningsområdet[2] [Boone 1999]. Mens problemområdet beskriver hva som særpreger applikasjonene mer generelt, beskriver løsningsdomenet hvordan noen konkrete eksempelapplikasjoner er blitt realisert tidligere [Boone 1999]. Informasjon om problemdomenet innhentes først og fremst ved litteraturstudier og konsultering med eksperter på området. Løsningsdomenet realiseres ved å fremskaffe (eller utvikle) noen *eksempelapplikasjoner* [Fayad et al. 1999, Bosch et al. 1999]. Med eksempelapplikasjon menes en implementert applikasjon, eller prototyp på applikasjon, som faller innenfor området til rammeverket.

Fayad hevder at det eksisterer relativt få aksepterte modeller for utvikling av rammeverk [Fayad et al. 1999]. De som finnes, kan kategoriseres grovt etter hvilken informasjon de baserer rammeverket på. Den ene kategorien er de som analyserer og abstraherer design fra eksisterende applikasjoner (løsningsområdet). Ved å foreta gjenbruk av generelle og relevante konstruksjonsløsninger fra disse, kan man utvikle rammeverket direkte uten dypere kunnskaper omkring dets område. Denne tilnærmingen benevnes som *refactoring* [Miller et al. 1999]. Den andre ser bort fra eksisterende designløsninger og konsentrerer seg heller om å studere problemområdet. Målet er å komme frem til en kravspesifikasjon for rammeverket. Denne benevnes som *a priori*, og er mer i tråd med ordinær applikasjonsutvikling [Miller et al. 1999, Sommerville 1998]. Rammeverket baseres her på den identifiserte kravspesifikasjonen. Det er verdt å understreke at man også kan studere eksisterende eksempelapplikasjoner ved en *a priori* tilnærming. Disse er også med på å beskrive problemområdet [Jacobson et al. 1999]. Man vil da se etter funksjonalitet fra "utsiden" av applikasjonene (som bruker), fremfor design og kildekode fra "innsiden". Om ikke fullt utviklede applikasjoner er tilgjengelige kan også prototyper anvendes [Jacobson et al. 1999].

---

[1] Den engelske betegnelsen på dette begrepet er "*framework domain"*.

[2] De engelske betegnelsene på disse begrepene er henholdsvis "*problem domain*", og "*solution domain*".

I denne oppgaven kunne det være fristende å anvende en refactoring-basert modell. Dette fordi jeg har utviklet to selvstendige applikasjoner under studiet. Hos disse har jeg tilgang til både design og kildekode. Jeg kunne da ha benyttet modeller som ”Systematic Generalization” [Schmid 1999], ”The Harvesting Process” [Boone 1999], eller en ”pattern”-modell av Roberts [Roberts et al. 1996] for å ekstrahere design. Likevel vil jeg argumentere for at dette er uheldig. Dette har to årsaker: For det første ønsker jeg å også foreta en kvalitativ studie av en rekke andre, eksterne, eksempelapplikasjoner i oppgaven (fire stykker). For disse har jeg verken tilgang til design eller kildekode (bare funksjonalitetsbeskrivelse sett fra et brukerperspektiv). Derfor kunne de heller ikke blitt anvendt i modellene nevnt over. For det andre vil jeg også vurdere mine egne eksempelapplikasjoner som prototyper. De er ikke fullt ut ferdigstilte applikasjoner, og derfor er det også unaturlig å forvente at konstruksjonen hos disse skal representere noen utprøvd og god løsning generelt. I hvert fall med det mål å basere et rammeverk på dem. Det som er interessant med disse, er snarere hva slags funksjonalitet de tilbyr og erfaringene jeg har gjort rundt dette, fremfor design.

I denne oppgaven har jeg valgt å anvende en *a priori* fremgangsmetode. Det vil si, jeg tar sikte på å utvikle rammeverket ved å analysere problemområdet. Informasjonen jeg vil basere analysen på, vil hovedsakelig bestå av erfaringer gjort fra utviklingen av Mocado og Mobitras, funksjonalitetsbeskrivelsen fra fire andre eksempelapplikasjoner, samt relativt mye litteratur på området ellers. I tillegg vil jeg rådføre og diskutere oppgaven underveis med mine veiledere hos SINTEF. Disse kan vurderes som eksperter på oppgavens område. Målet med analysen er å komme frem til en kravspesifikasjon for et rammeverk. Denne vil bli presentert i Kapittel 9.

2.3 Oppsummering

Jeg har i dette kapitlet sett nærmere på rammeverk. Jeg har drøftet sentrale begreper som black box og white box rammeverk, kjernerammeverk og inkrementer, og horisontale og vertikale rammeverk. Ikke minst har jeg også belyst hva jeg i oppgaven definerer til å være rammeverk. Videre har jeg drøftet utvikling av rammeverk i forhold til ordinær applikasjonsutvikling, og belyst forskjellen mellom abstraction og a priori tilnærming. Jeg gjorde det også klart at jeg i denne oppgaven vil nærme meg rammeverket ved å analysere problemområdet, og vil derfor anvende en a priori fremgangsmetode.

Kapittel 3

Kontekst og kontekstoppmerksomhet

> *"Mari og Petter er på café. De har bestilt litt mat, og sitter og spiser. De verken snakker sammen eller ser på hverandre. Når Petter spør om hun kan sende ham saltet, fører hun bøssen bestemt bort uten å løfte blikket."*

Når mennesker kommuniserer blir mye av informasjonen bestemt av ytre omstendigheter [Bjartveit et al. 1996]. Hva som sies eller gjøres må alltid tolkes i lys av den kontekstuelle situasjonen aktørene befinner seg i. En kommentar mellom to venner kan bety noe helt annet sagt mellom to fiender. I eksempelet med Mari og Petter har man liten mulighet til å forstå hva som skjer når ikke situasjonen er kjent på forhånd. Er de sinte på hverandre, eller er de triste? Er det nødvendigvis noe i det hele tatt? Selv om de verken ser på hverandre eller sier noe, kommuniserer de fortsatt aktivt. Det er umulig for mennesker å "ikke-kommunisere" [Bjartveit et al. 1996].

Når mennesker jobber med datamaskiner benytter maskinene sjeldent kontekstuell informasjon for å bestemme innholdet i interaksjonen [Dey et al. 1999b, Pascoe 1998]. Hvis man foretar et menyvalg i en applikasjon, eller markerer en tekst, har dette sin helt spesielle betydning og fører som regel til en forutsigbar handling. Denne er uavhengig av brukerens situasjon. Dey hevder at man kan heve nivået på kommunikasjonen mellom applikasjonen og brukeren ved å øke applikasjonenes tilgang til brukerens kontekst [Dey et al. 1999a]. Dette er også noe av hva man søker å oppnå med kontekstoppmerksomme applikasjoner.

I dette kapitlet vil jeg presentere og gjøre rede for sentrale begreper som anvendes i oppgaven. Jeg vil begynne med å drøfte kontekst. Jeg vil undersøke hva det er, og hvilke kategorier det kan deles inn i. Jeg vil drøfte kontekstoppmerksomhet og kontekstoppmerksomme applikasjoner, og relevante kategorier rundt dette. En av disse kategoriene, presentasjon av kontekstrelevant informasjon, er også et fokuspunkt i oppgaven. Jeg vil derfor drøfte denne spesielt. Til sist vil jeg

drøfte mobilitetsbaserte applikasjoner, og hvordan kontekstoppmerksomhet anvendes blant disse.

3.1 Situasjon og kontekst

Jeg vil begynne med å se på en relativt generell definisjon på kontekst. Selv om denne vil endres på senere, i henhold til oppgavens område, vil jeg innlede med denne som utgangspunkt. Definisjonen er hentet fra Oxford Advanced Learners Dictionary, og lyder som følger:

> *"Context: circumstances in which situations happend or in which situations is to be considered"*
>
> -[OALD 1990, side 254]

En situasjon i denne definisjonen kan være så mye. Sitter man og leser en bok, er man i en situasjon. Værforholdene er en situasjon. To mennesker som møtes på en café er en situasjon. Ofte kan det være vanskeligere å avgjøre hva som *ikke* er en situasjon. I hvert fall hva som ikke er en interessant situasjon. Det som er interessant for noen kan være uinteressant for andre. I tillegg kan det også variere hva man oppfatter som selve situasjonen, og hva som isteden er dens kontekst. La oss ta et eksempel der to biler har kollidert. Politiet har ankommet og prater med de involverte. En tilfeldig passerende vil kanskje mene at kollisjonen sammen med politiet er selve situasjonen. Stedet og menneskene rundt blir oppfattet som situasjonens kontekst. Politiet, derimot, vil kanskje mene at kollisjonen alene er situasjonen, mens sted, tidspunktet, hendelsesforløpet før kollisjonen, og lignende, er dens kontekst. Fremfor at det ene er mer riktig enn det andre, viser det seg snarere at svaret er individuelt. Kontekst er et subjektivt konsept som defineres av hver enkelt observatør [Pascoe 1998]. Hva som er interessant ved en situasjon avhenger isteden av hva informasjonen skal anvendes til. Det samme gjelder også for kontekstoppmerksomme applikasjoner Det som varierer mellom disse, er hva som vurderes som en interessant situasjon, og hva som videre er interessant i situasjonens omstendigheter [Dey 2000].

Den første avgrensingen jeg vil foreta, er å bestemme hvilke situasjoner som er interessante i denne oppgaven. Mer spesifikt vil dette si: *Hvem* eller *hva* skal applikasjonene observere. Svaret er PDA-brukeren. Det er hans situasjon som er relevant. Det er han applikasjonene skal observere og forholde seg til. Likevel vil jeg også nevne andre mulige objekter. Det finnes for eksempel applikasjoner som er oppmerksomme på sin egen kontekst. Her kan kontekstuell informasjon være tilgjengelige nettverksressurser som for eksempel båndbredde og printere, sammen med deres assosierte tjenestekvalitet [Dix et al. 2000]. Andre applikasjoner kan isteden ha sin oppmerksomhet rettet mot PDA-en der applikasjonen eksekverer. Batterikapasitet kan vurderes som kontekstuell informasjon her. Men

denne oppgaven skal som sagt dreie seg om brukeren. Dette vil ikke nødvendigvis utelukke for eksempel tilgjengelige printere som betydningsfull kontekst. Dette kan vurderes som viktig informasjon for å forklare også hans situasjon. Avgrensingen er mer en rettesnor i forhold til hvilke applikasjoner og kilder jeg vil studere og forholde meg til igjennom resten av oppgaven.

Schilit argumenterer for at det spesielt er tre typer informasjon som er relevant for å beskrive en brukers kontekst: Hvor han er, hvem han er sammen med, og hvilke ressurser han er i nærheten av [Schilit et al. 1994]. Med det siste mener han tilgjengelig nettverksoppkoblet utstyr som kan anvendes av PDA-en (som for eksempel en printer). Dey argumenterer for at kontekst ikke kan defineres på denne måten [Dey 2000]. Kontekst kan ikke defineres ved å ramse opp alle mulige elementer den kan inneholde. Dette vil, som nevnt, variere fra applikasjon til applikasjon. Kontekstoppmerksomme applikasjoner konsentrerer seg heller om å observere bestemte entiteter, og stiller spørsmålene hvem, hva, hvor, og når rundt disse. Denne informasjonen anvendes til aktivering av ønsket funksjonalitet. Dey definerer av den grunn kontekst slik:

> *"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves."*
>
> – [Dey 2000, side 4]

Denne definisjonen vil også jeg anvende. Den mest vesentlige forskjellen i forhold til den første, er at kontekst her også beskriver selve situasjonen, ikke bare dens omstendigheter. Dette er analogt med at situasjonen til en bruker både beskriver han selv, i tillegg til hans omgivelser. Igjen vil jeg presisere at entitetene jeg vil konsentrere meg om er brukerne, ikke steder eller andre objekter som definisjonen også inkluderer. Det er brukernes situasjon som er interessant. Likevel gir definisjonen rom for tolkning. Hva slags informasjon tilhører egentlig en persons (entitets) situasjon? Inkluderer den for eksempel informasjon om personen selv, som navn, adresse, og lignende? Dey hevder selv at svaret på dette er "ja" [Dey et al. 1998]. Identitet, interesser, preferanser, og lignende, vurderes generelt for å være en del av brukerens kontekst [Dey et al. 1998, Lieberman et al. 2000, Salber et al. 1998]. Slik vil også jeg anvende kontekstbegrepet i denne oppgaven.

3.1.1 Kategoriseringer av kontekst

Selv om ikke kontekst kan defineres ved eksempler [Dey 2000], er det likevel hensiktsmessig å kategorisere informasjonen etter hva den beskriver. Dette vil heve nivået på funksjonalitetsanalysen senere i oppgaven (jamfør Kapittel 7 og

Kapittel 8). Det viser seg videre at man kan konstruere en rekke slike kategorier. Blant de mest sentrale er: *Fysisk kontekst*, *systemkontekst*, *applikasjonskontekst*, *aktivitetskontekst*, *emosjonellkontekst*, *sosialkontekst*, *historikkontekst*, *tidskontekst*, og *identitetskontekst*. Jeg vil gi en kort forklaring på disse under.

Mange utnevner Fysisk kontekst som den mest sentrale kategorien kontekst kan deles inn i [Salber et al. 1998, Dey et al. 1998, Brown 1995]. Denne kategorien innholder fysiske parametere som blant annet lokasjon, temperatur, puls, lyd-/lysnivå, nære gjenstander, og lignende. Dette er typisk målbare verdier som kan innhentes ved hjelp av sensorer. Blant kontekstoppmerksomme applikasjoner er kontekst fra denne kategorien mest anvendt [Salber et al. 1998].

Som kategori nummer to kommer systemkontekst [Salber et al. 1998] (også benevnt infrastrukturkontekst [Dix et al. 2000]). Denne inkluderer båndbredde, nettverksbelastning, nettverksressurser, operativsystem, og lignende. Dette er informasjon som beskriver omgivelsene til applikasjonen. Derfor kan den også betraktes som akkurat det; applikasjonens kontekst fremfor brukerens. Likefullt kan den også være relevant for å beskrive brukerens situasjon. Jeg har allerede nevnt tilgjengelige printere som en nettverksressurs. En tilgjengelig printer kan også vurderes som en nær gjenstand. Dette vil i så fall være informasjon som tilhører kategorien ”fysisk kontekst”. Dette illustrerer at kategoriene jeg presenterer her ikke nødvendigvis er distinkte mengder, men gjerne kan overlappe.

En tredje kategori benevnes som applikasjonskontekst [Salber et al. 1998]. Dette er ikke applikasjonens egen kontekst, som navnet kan villede en til å tro. Imidlertid er det informasjon som beskriver brukerens interaksjon med applikasjonen. Om han for eksempel til stadighet anvender et spesielt menyvalg i applikasjonen, kan dette vurderes som en del av hans applikasjonskontekst [Dey et al. 1998]. I nyere versjoner av operativsystemet ”Windows”, for eksempel, vil slike repeterende menyvalg fremheves spesielt i applikasjonen. Mindre brukt funksjonalitet vil isteden skjules. Disse er med andre ord oppmerksomme til brukerens applikasjonskontekst. Abowd nevner også *informational contekst* som en egen kategori [Abowd 1998]. Denne inneholder informasjon om hvilke data brukeren jobber med i en applikasjon. Jeg vurderer imidlertid denne kategorien som en underkategori av applikasjonskonteksten.

Aktivitetskonteksten tilhører kategoriene med informasjon som vanskelig lar seg registrere automatisk (som for eksempel ved hjelp av sensorer). Her beskriver informasjonen brukerens gjøremål [Dey et al. 1999a]. Er han på ferie eller jobb? Sitter han i et møte, eller spiser han lunsj? Er han ute og trener, eller løper han bare etter bussen? Ofte er brukeren selv den eneste rette til å besvare disse spørsmålene. En variant av aktivitetskonteksten er kategorien *intentional context* [Abowd 1998]. Fremfor å beskrive hvilke aktiviteter brukeren faktisk holder på med, inneholder denne informasjon om hva slags aktiviteter *han kunne tenkt seg* å drive med.

En kategori som ligner aktivitetskonteksten med tanke på registrering, er emosjonellkontekst [Abowd 1998]. Denne beskriver humøret til brukeren. Er han trist og lei, eller glad og fornøyd? Er han sliten og trøtt, eller opplagt og frisk? Hva en

bruker har av følelser innerst inne kan vanskelig måles ved hjelp av en sensor. Noe kan likevel utledes/gjettes på ved å for eksempel studere adferdsmønster, lokasjon, eller lignende.

En sjette kategori er brukerens sosiale kontekst [Salber et al. 1998]. Dette er informasjon som beskriver hvilke mennesker brukeren omgir seg med. Hvem er i nærheten, og kjenner han noen av disse? Hva er deres kontekst? Salber beskriver sosial kontekst som *andres* identitet, aktivitet, personlighet, og lignende [Salber et al. 1998]. Slik informasjon benytter han blant annet i en applikasjon med navnet "*Are You Reading Me*". Her blir opplysninger om en annen persons e-postkasse anvendt som sosial kontekst. Denne informasjonen er med på å avgjøre om det har noen hensikt å sende en e-post til denne personen dersom raskt svar er av interesse. Er hans postkasse for full, kan det være mer hensiktsmessig å bruke telefonen. Mottagerens kontekst (postkassens status) blir dermed viktig sosial kontekst for avsenderen.

Historisk kontekst blir av flere vurdert som viktig kontekstuell informasjon [Lieberman et al. 2000, Abowd 1998, Dey et al. 1998]. Dette er situasjoner som brukeren har befunnet seg i tidligere, og som anvendes for å beskrive hans situasjon i nåtid. Spesielt det siste er viktig. Informasjonen må anvendes for å beskrive hans situasjon i nåtid. I motsatt fall vil informasjonen bare tjene som konteksthistorie (fremfor historisk kontekst). Denne kan isteden tenkes på som en logg for å bevare informasjon om hvilke situasjoner brukeren har befunnet seg i tidligere. Innholdet her trenger med andre ord ikke nødvendigvis anvendes for å tolke brukerens situasjon i nåtid. Men om en applikasjon anvender det slik, så vil informasjonen falle innunder kategorien "historisk kontekst" hos denne.

Den åttende kategorien, tidskonteksten [Chen et al. 2000], inneholder bare tid og dato. De fleste kildene jeg har anvendt vurderer disse parametrene som relevante for å beskrive en brukers situasjon [Chen et al. 2000, Dey et al. 1998, Brown 1995, Pascoe 1998].

Som allerede drøftet blir brukerens identitetskontekst vurdert som kontekstuell informasjon [Dey et al. 1998, Lieberman et al. 2000, Salber et al. 1998]. Dette er informasjon som beskriver hvem brukeren er, som hans navn, adresse, telefonnummer, og lignende. Jeg har også valgt å definere en underkategori av denne kategorien benevnt *preferanse-, og interessekontekst*. Dette er informasjon som beskriver hva brukeren liker å holde på med, eller hvordan han liker å ha det. Hobbyer, matvaner, klesstil, og lignende vil komme innunder denne underkategorien.

Kontekst kan også kategoriseres etter andre forhold enn det som er gjort over. For eksempel argumenterer Chen for at man kan dele den opp i en primær og en sekundær kategori [Chen et al. 2000]. Primærkontekst[1] inkluderer lokasjon, entitet, aktivitet, og tid, og fungerer som grunnparametere for å bestemme også annen kontekstuell informasjon. En students lokasjon, tid, og timeplan kan for eksempel benyttes til å finne ut av hvilken forelesning han befinner seg på (hans

[1] Den samme kategorien benevnes av Dix som nøkkelkontekst [Dix et al. 2000].

aktivitetskontekst). Videre kan slik informasjon også deles inn i kategoriene lavnivå og høynivå [Chen et al. 2000]. Det første nivået er enkle, ikke-sammensatte parametere som gjerne kan registreres ved hjelp av sensorer, mens det andre er en mer diffus og ikke direkte målbar form for informasjon. Brukerens humør er et eksempel på høynivå informasjon (hans emosjonelle kontekst). Ergo vil lavnivåkontekst i mange tilfeller også være primærkontekst, mens høynivåkontekst vil være sekundærkontekst.

3.2 Kontekstoppmerksomme applikasjoner

Kontekstoppmerksomme applikasjoner er strengt tatt ikke noe nytt. Mye elektronisk utstyr som vi omgir oss med i det daglige, har tatt i bruk denne teknologien lenge. Dette gjelder for eksempel automatiske døråpnere, selvfokuserende fotografiapparater, sensorstyrte pissoarer, og lignende. Kontekstoppmerksomhet er med andre ord et vanlig krav i mange systemer allerede, og vil trolig også få en større rolle hos fremtidige PDA-applikasjoner [Pascoe 1998]. Pascoe definerer en applikasjon som kontekstoppmerksom[1] dersom den har muligheten til å registrere forskjellige tilstander rundt seg selv og sine omgivelser [Pascoe 1998]. Denne definisjonen inkluderer imidlertid kun konteksten til applikasjonen selv, og ikke konteksten til brukeren. Lieberman, derimot, definerer kontekstoppmerksomhet til å være applikasjoner som registrerer kontekst og modifiserer seg selv deretter, uten å motta eksplisitt hjelp fra brukeren [Lieberman et al. 2000]. Hensikten er at brukeren skal slippe å taste inn slik informasjon selv. Denne definisjonen vil også inkludere oppmerksomhet i forhold til brukerens kontekst. Likevel er den svært spesifikk. Den forlanger at konteksten som anvendes må benyttes til modifisering av funksjonalitet. Jeg vil isteden anvende en definisjon fremsatt av Dey:

> *"A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task"*
>
> – [Dey 2000, side 6].

Denne definisjonen inkluderer både oppmerksomhet i forhold til brukerens kontekst, og tillater flere ulike anvendelsesområder. Imidlertid vil jeg foreta en avgrensning. Fremfor å studere enhver applikasjon som faller innunder denne definisjonen, vil jeg konsentrere meg om en delmengde. Jeg vil fokusere på

[1] Pascoe benevner kontekstoppmerksomhet som "*context-sensitivity*" [Pascoe 1998]. Også "*context-aware*", "*adaptive*", "*situated*", "*environment-directed*", med mer, blir anvendt som synonymer [Dey 2000]. Jeg har valgt benevnelsen "kontekstoppmerksom" da mye av arbeidet i denne oppgaven er inspirert av Dey, som anvender betegnelsen "Context-Aware" [Dey 2000].

applikasjoner som anvender kontekstoppmerksomhet for å tilby relevant informasjon til brukeren. Med relevant informasjon mener jeg enten brukerens kontekst direkte, eller informasjon som er relevant på grunnlag av denne. Det første definerer Pascoe til å falle innenfor kategorien *contextual sensing*, mens det siste defineres som *contextual augmentation* [Pascoe 1998]. Contextual sensing er med andre ord fremvisning av brukerens kontekst direkte. Opptegning av hans posisjon på et kart er et eksempel på slik funksjonalitet. Contextual augmentation er isteden fremvisning av informasjon knyttet opp mot en kontekst (assosiering av data til kontekst). Opplysninger om en severdighet i nærheten av brukeren er et eksempel her. Pascoe definerer også to andre utfyllende kategorier: *contextual adaption*, og *contextual resource discovery* [Pascoe 1998]. Det første er funksjonalitet som endrer eller tilpasser seg automatisk i forhold til brukerens kontekst. Et menyvalg i en applikasjon med etiketten ”tilbudsvarer” kan for eksempel produsere to helt forskjellige resultater eksekvert i to forskjellige butikker. Den andre, contextual resource discovery, er funksjonalitet for å lete frem og anvende tilgjenglige, kontekstrelevante, tjenester. Om brukeren ønsker å skrive ut et dokument fra sin PDA, kan slik funksjonalitet resultere i at nærmeste printer bestandig blir valgt.

Også Schilit og Dey har kommet frem til lignende kategorier som Pascoe [Schilit et al. 1994, Dey 2000]. Dey argumenterer blant annet for at Pascoes contextual resource discovery ikke fortjener en egen kategori, men heller bør vurderes som en kombinasjon av contextual sensing og contextual adaption. Anvendelse av nære printere vil for eksempel si at contextual sensing anvendes for å lete opp tilgjengelige printere (brukerens systemkontekst), for å deretter utføre contextual adaption når brukeren skriver ut et dokument (tilpasning av utskriftsfunksjonalitet). Dey benytter seg isteden av følgende tre kategorier [Dey 2000]:

1. Presentasjon av informasjon og tjenester til en bruker.
2. Automatisk eksekvering av en tjeneste
3. Assosiering av informasjon til kontekst.

Med utgangspunkt i disse kategoriene vil jeg med andre ord konsentrere meg om kontekstoppmerksomme applikasjoner som anvender og presenterer kontekst (1), og som anvender og presenterer informasjon assosiert til en kontekst (3).

3.3 Kontekstrelevant informasjon

Nedenfor vil jeg benevne informasjon som er assosiert til en kontekst, for kontekstrelevant informasjon. Et eksempel på slik informasjon er avtaler man fører

inn i digitale kalendere, som for eksempel Microsoft Outlook[1]. Et møte her kan vurderes som informasjon assosiert til et tidspunkt. Når brukeren befinner seg i den gitte kontekstuelle situasjonen (tidspunktet for møtet, eller litt i forveien), vil informasjonen presenteres som en påminnelse.

En underkategori av kontekstrelevant informasjon er informasjon som er *kontinuerlig*. Dette vil si at informasjonen varierer med brukerens kontekst [Brown 1998]. Hvis konteksten endrer seg, endrer også informasjonen seg. Et eksempel på dette er applikasjoner som presenterer avstanden mellom brukeren og et gitt sted. Når brukeren flytter på seg, vil også avstanden endre seg. Det motsatte, *diskontinuerlig* informasjon, forholder seg isteden statisk [Brown 1998]. Slik informasjon er fortsatt assosiert til en kontekst, men innholdet avhenger ikke av konteksten. Eksempelet over, med møteavtalen, er et slikt tilfelle. Innholdet i avtalen er helt uavhengig av tidspunktet det blir lest (hans tidskontekst).

I denne oppgaven vil jeg konsentrere seg om diskontinuerlig kontekstrelevant informasjon. Applikasjonene jeg vil fokusere på vil med andre ord kun anvende kontekst for å presentere statisk informasjon. Likevel vil jeg ikke utelukke kontinuerlig informasjon helt. Grunnen til dette er at slik informasjon i mange tilfeller også kan oppfattes som kontekstuell informasjon. Brukerens avstand til ett eller annet objekt, er for eksempel en del av hans fysiske kontekst. Dermed er dette også funksjonalitet som er inneholdt i kategorien ”presentasjon av informasjon og tjenester til en bruker” [Dey 2000]. Denne er, som nevnt, også sentral i oppgaven. Nedenfor vil jeg uansett benevne dette som presentasjon av kontekst, fremfor presentasjon av kontinuerlig informasjon.

Jeg vil tillegge kontekstrelevant informasjon en rekke egenskaper. Først og fremst kan slik informasjon vurderes som *gyldig* eller *ugyldig*. At en bestemt mengde[2] informasjon betraktes som gyldig, vil si at applikasjonen som anvender den finner den relevant i forhold til brukerens situasjon. Vanligvis avgjøres dette ved å se på informasjonens assosierte kontekst i forhold til brukerens kontekst. Om brukeren befinner seg i samme situasjon som informasjonen, vil informasjonen for han fremstå som relevant [Pascoe 1998].

Når en bestemt mengde informasjon blir vurdert som gyldig, kan den *aktiveres*. Dette betyr at informasjonen kan presenteres for brukeren. Den er da relevant i forhold hans situasjon. Når informasjonen ikke lenger kan vurderes som relevant (den er blitt ugyldig), kan den *deaktiveres*. Dette betyr at den ikke lenger skal presenteres for brukeren. Jeg vil understreke ”kan”, siden det blir opp til hver enkelt applikasjon hvordan informasjonen faktisk skal benyttes. Det samme gjelder også hva slags algoritme som skal anvendes for å avgjøre informasjonens gyldighet.

Selve prosessen en applikasjon gjennomgår når den leter frem gyldig informasjon som skal aktiveres, vil jeg benevne som dens *aktiveringsprosess*. Alle appli-

---

[1] Se http://www.microsoft.com/office/outlook/ (dato: 29.07.2002).

[2] Med ”bestemt mengde” mener jeg avgrenset, bestemt informasjon. Møteavtalen er et eksempel på slik informasjon.

kasjonene jeg har sett på i denne oppgaven, har et slikt system. Brown, for eksempel, benevner dette i *Stick-e notes* for *triggering engine* [Brown 1998] (*Stick-e notes* er en arkitektur jeg vil komme tilbake til senere, jamfør Kapittel 10.2). Denne "motoren" traverserer kontinuerlig gjennom all kontekstrelevant informasjon som applikasjonen oppbevarer, og kontrollerer den opp mot brukerens kontekst. Informasjon som blir funnet gyldig, blir aktivert og presentert [Brown 1998]. I denne oppgaven skal imidlertid "aktiveringsprosessen" kun forstås som en abstraksjon. Den henviser ikke til noen spesiell implementasjon.

Kontekstrelevant informasjon har oppsummert en assosiert kontekst som beskriver når informasjonen den oppbevarer er gyldig. Siden denne konteksten benyttes under aktiveringsprosessen, vil jeg ofte benevne den som *aktiveringskonteksten*. Bestemte mengder kontekstrelevant informasjon har med andre ord hver for seg en assosiert aktiveringskontekst.

3.4 Mobilitetsbaserte kontekstoppmerksomme applikasjoner

Kontekstoppmerksomhet kan anvendes innen flere ulike applikasjonsområder, og til flere ulike formål. Lieberman hevder at slik funksjonalitet blant annet er selve nøkkelen for å utvikle gode applikasjoner innenfor området "kunstig intelligens" [Lieberman et al. 2000]. Videre nevner han også modifisering av brukergrensesnitt som et annet relevant område [Lieberman et al. 2000]. Denne oppgaven skal imidlertid dreie seg om en helt annen kategori, nemlig kontekstoppmerksomhet blant *mobilitetsbaserte applikasjoner*. Jeg vil forklare slike applikasjoner nærmere nedenfor.

Jeg vil begynne med å drøfte begrepet *mobilitet*. Dix hevder at et grunnleggende aspekt ved mobile gjenstander, er at de av natur ofte endrer lokasjon [Dix et al. 2000]. Som nevnt er gjenstanden av interesse i denne oppgaven, brukeren selv. Det er muligheten han har for å være mobil som gjør kontekstoppmerksomhet interessant her. Det er hans kontekstuelle situasjon applikasjonene ønsker å tilby relevant informasjon i henhold til. Imidlertid finnes det flere grader av mobilitet. Hvor mobil en PDA-bruker i realiteten er, avhenger både av PDA-ene og applikasjonene han anvender [Thanh 2001]. Hvis hans PDA ikke kan endre lokasjon uten å miste eller tape kvalitet på sine tjenester (som for eksempel nettverksforbindelsen), vil dette også kunne begrense hans egen mobilitet[1] [Thanh 2001].

Applikasjoner kan også kategoriseres i forhold til hvor bevisste de er på mobilitet. Den første kategorien er de som er *mobilitetsubevisste* [Thanh 2001]. Disse har ikke noe forhold til mobilitet i det hele tatt. Ofte kan det være like greit å la applikasjonene slippe å ta hensyn til dette. Om de skal oppføre seg nøyaktig likt uavhengig av mobilitet, kan slike forhold isteden gjøres transparent i et underliggende lag (som for eksempel i operativsystemet).

---

[1] Dette benevnes som brukerens "terminalmobilitet" [Thanh 2001].

Den andre kategorien av applikasjoner, er de *mobilitetsbevisste* [Thanh 2001]. Som navnet henspeiler, er disse applikasjonene oppmerksomme på at de vil delta i en mobil situasjon. Enten brukeren, terminalen, eller applikasjonen kan oppleve mobilitet [Thanh 2001]. Ved å legge inn eksplisitt funksjonalitet i disse applikasjonene slik at de tilpasses mobile forhold, kan tjenestene som tilbys også gjøres mobilitetstransparente. Dette vil si at applikasjonen jobber aktivt for å skjule mobile forhold ovenfor brukeren. Dette er for eksempel et vanlig mål ved utvikling av tjenester for mobiltelefoner [Thanh 2001].

Kategorien jeg vil rette søkelyset på, er de *mobilitetsbaserte*. Dette er applikasjoner som har basert sine tjenester på at brukeren er mobil [Thanh 2001]. De anvender med andre ord bevisstheten om mobilitet eksplisitt for å skape nye tjenester. Av den grunn er de også mobilitetsbevisste. En applikasjon som for eksempel presenterer for brukeren hvor han befinner seg på et kart, er en slik applikasjon. Her anvendes informasjon om brukerens mobilitet for å skape en egen selvstendig tjeneste. Applikasjonene er andre ord mer enn bare bevisste på brukerens mobilitet; de er konstruert med dette som utgangspunkt.

Applikasjonene jeg har valgt å se på i denne oppgaven, anvender kontekstoppmerksomhet for å registrere brukerens situasjon. Når brukeren er mobil, kan imidlertid denne situasjon endre seg ofte [Dey et al. 1999a]. Nye gjenstander i omgivelsene vil komme til, sosiale situasjoner (som møtevirksomhet eller cafébesøk) vil dannes og splittes, og aktivitetene som bedrives vil endre seg hyppigere. Dix, med flere, mener at lokasjon er den absolutt viktigste kontekstuelle parameteren mobilitetsbaserte applikasjoner har behov for å kjenne til [Dix et al. 2000, Salber et al. 1998, Chen et al. 2000]. Denne parameteren kan fungere som en slags indeksnøkkel for å registrere annet kontekstuell informasjon [Dix et al. 2000, Chen et al. 2000]. Kjenner man for eksempel flere personers lokasjon hver for seg, kan man også avgjøre hvem som er i nærheten av hverandre (sosial kontekst). Hos applikasjonene jeg vil studere, vil lokasjon med andre ord være en særskilt sentral kontekstparameter.

En bruker kan endre lokasjon på flere ulike måter. Skifter man mappefokus i en filutforsker, endrer man også lokasjon [Dix et al. 2000]. Det samme er tilfellet når man navigerer mellom ulike sider på Internett [Dix et al. 2000]. Dette reflekteres også på måten man omtaler det på; ”gå til en spesiell adresse”, ”gå tilbake”, ”bla oppover”. Det som varierer, er referansesystemet som benyttes. Man skiller i så måte mellom den *fysiske verdenen*, og diverse andre *virtuelle verdener* [Dix et al. 2000, Pascoe 1998]. I eksempelet ovenfor var Internett en slik virtuell verden [Dix et al. 2000]. Felles for referansesystemene er at de vanligvis har et forhold til lokasjon og nærhet [Dix et al. 2000]. I denne oppgaven vil jeg imidlertid se mest på den fysiske verdenen. Dette er en den ikke-digitale virkeligheten vi alle befinner oss i hele tiden. Det er brukerens mobilitet her som er interessant. Jeg vil derfor oppsummert undersøke hvordan kontekstoppmerksomhet kan anvendes for å tilby kontekstuell-, og kontekstrelevant informasjon i forhold til brukerens mobilitet i den fysiske verdenen.

3.5 Oppsummering

I dette kapitelet har jeg gjennomgått sentrale begreper for oppgaven. Jeg begynte med begrepet ”kontekst”, og drøftet hvordan dette kan defineres. Jeg belyste hvordan det kan være relevant å snakke om kontekst for mange ulike entiteter, men at jeg vil fokusere på PDA-brukerens kontekst. Jeg presenterte også noen ulike kategorier for hvordan slik informasjon kan struktureres.

Et annet sentralt begrep i oppgaven er kontekstoppmerksomme applikasjoner. Jeg introduserte noen divergerende definisjoner på hva dette var, og presenterte noen ulike anvendelsesområder for kontekstoppmerksomhet. Ett av disse områdene var assosiering av informasjon til kontekst, noe jeg benevnte jeg som ”kontekstrelevant informasjon”. Jeg belyste også forskjellen mellom kontinuerlig og diskontinuerlig informasjon, og understreket at jeg hovedsakelig vil fokusere på det siste. Jeg beskrev videre hvordan slik informasjon kunne betraktes som enten gyldig eller ugyldig. Informasjonen som ble vurdert som gyldig, kunne også aktiveres (og senere deaktiveres). Dette ble av en applikasjon avgjort i det jeg benevnte som ”aktiveringsprosessen”.

Til sist presenterte og drøftet jeg mobilitetsbaserte applikasjoner. Jeg belyste hva det vil si at man er mobil, og noen faktorer som bestemmer graden av mobilitet i en informatikksammenheng. Jeg presenterte tre ulike kategorier for å bestemme applikasjoners bevissthet om mobilitet, og trakk frem kategorien ”mobilitetsbaserte applikasjoner” som sentral for denne oppgaven. Spesielt viktig for mobilitetsbaserte applikasjoner var imidlertid kontekstparameteren ”lokasjon” innenfor den fysiske verdenen.

Kapittel 4

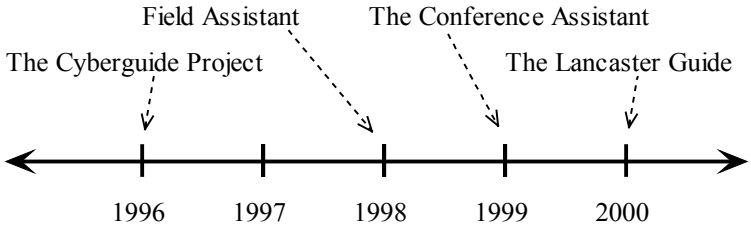
Fire eksterne eksempelapplikasjoner

Hensikten med dette studiet er å utvikle et rammeverk. For å oppnå dette, trenger jeg en forståelse for hvordan applikasjonene rammeverket søker å støtte, virker. Med dette mener jeg hvordan de anvender kontekstoppmerksomhet sammen med kontekstrelevant informasjon. Jeg vil forsøke å foreta en slik analyse i Kapittel 7 og 8. For å danne et godt utgangspunkt for denne fasen, har jeg forsøkt å tilegne meg førstehåndskunnskap om rammeverkets område. Jeg har blant annet utviklet to applikasjoner som et slikt rammeverk kunne ha blitt benyttet for å støtte. Likevel ønsker jeg ikke at analysen i Kapittel 7 og 8 bare skal baseres på erfaringene gjort fra disse. Grunnen til dette er at det er jeg som har utviklet begge to, slik at de av den grunn kan ha blitt litt for like. Derfor har jeg også valgt å studere en rekke andre eksempelapplikasjoner. For at leseren lettere skal kunne følge mine argumenter og avgjørelser utover i oppgaven, gir jeg en kort presentasjon over de utvalgte her. Selve analysen vil jeg imidlertid presentere samlet i Kapittel 7 og 8, sammen med resultatene fra Mocado og Mobitras.

Ettersom fagfeltet oppgaven omhandler er relativt ungt[1], har antall tilgjengelige eksempelapplikasjoner vært begrenset. Enkelte hevder også at problemene rundt det å benytte sensorer har hindret mange applikasjonsutviklere fra å benytte kontekstoppmerksomhet [Dey 2000, Chen et al. 2000]. Applikasjonene jeg har valgt å studere plasserer seg likevel godt innenfor oppgavens område; de benytter alle kontekstoppmerksomhet sammen med kontekstrelevant informasjon. Jeg har imidlertid vektlagt at de bør dekke noe ulike anvendelsesområder. Mens noen av applikasjonene er ment for arbeidslivet, dreier andre seg om fritid. Likevel er det en liten overvekt av turistapplikasjoner. Dette skyldes at dette feltet representerer et særdeles relevant anvendelsesområde for denne typen applikasjoner [Long et al. 1996].

[1] Det første publiserte eksempelet på en mobilitetsbasert kontekstoppmerksom applikasjon er blitt hevdet å være The Active Badge Project fra Olivetti Research på slutten av 1980-årene [Want et al. 2001].

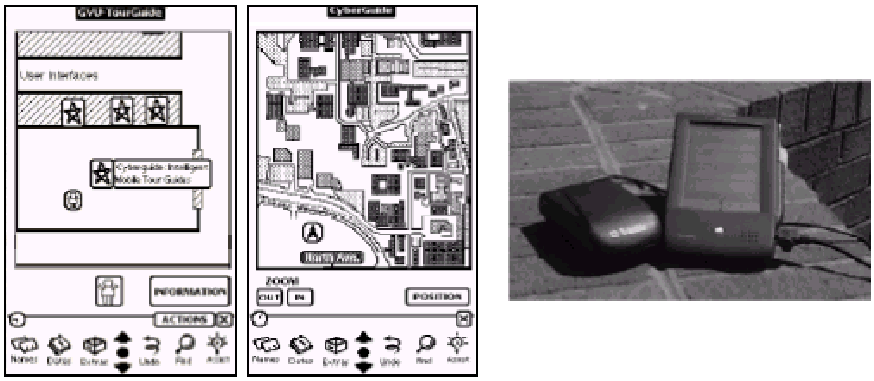
Applikasjonene jeg har valgt å studere er: *Cyberguide, Field assistant*, *The Conference assistant*, og *The Lancaster Guide* (se Figur 2).



Figur 2: Tidslinje for eksempelapplikasjoner – Tidslinjen viser årstallene for når eksempelapplikasjonene jeg har studert ble utviklet.

4.1 Cyberguide

Hos Georgia Institute of Technology har de hatt et prosjekt innen mobil informatikk benevnt "The Cyberguide Project" [Abowd et al. 1996, Long et al. 1996]. Prosjektet har bestått i å utvikle en rekke turistapplikasjoner for å undersøke hvordan besøkende hos The GVU Center Lab (på samme institutt) kunne bli støttet ved hjelp av lokasjonsbestemte tjenester. Utstyrt med en Apple MessagePad (se Figur 3) hadde de besøkende anledning til på egen hånd å ta seg frem på senteret og motta informasjon om attraksjonene de gikk bort til. En slik attraksjon kunne for eksempel være et annet utstilt prosjekt som pågikk på instituttet. For å la en applikasjon registrere brukerens lokasjon, ble et cellebasert system bestående av vanlige fjernsynskontroller hengende i taket, benyttet. Ved hjelp av infrarøde signaler fra disse, kunne PDA-ene avsløre sin egen (og dermed brukerens) lokasjon.



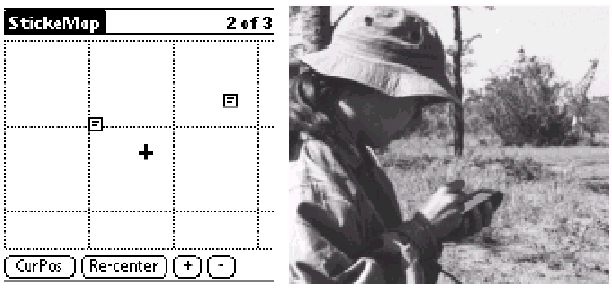
Figur 3: Cyberguide – Bildet til venstre viser innendørsversjonen, mens bildet i midten viser utendørsversjonen. Til høyre vises en Apple MessagePad med en tilkoblet GPS-mottaker. (Bildene er hentet fra [Abowd et al. 1996]).

For å presentere kontekstrelevant informasjon, ble brukerens lokasjon og synsretning registrert. Disse opplysningene ble benyttet til å vise brukeren hvor på kartet han befant seg, og til å finne frem informasjon om attraksjonene den antok at han så på. For å avgjøre synsretning, benyttet de en konteksthistorie over hans posisjonering. Denne avslørte i hvilken retning han kom fra, og dermed også i hvilken retning han antakeligvis så.

Enkelte av applikasjonene var også konstruert for utendørs bruk. I disse ble brukerens posisjon i tillegg bestemt av en tilkoblet GPS-mottaker, slik at han fritt kunne bevege seg inn og ut av senteret under besøket. I disse versjonene fikk han også mulighet til å "feste" sine egne notater til attraksjoner han besøkte.

4.2 Field assistant

Som en del av prosjektet "Mobile Computing in Fieldwork Environments" på Universitetet i Kent, ble det utviklet en rekke applikasjoner som sammen gikk under navnet Field Assistant Software [Pascoe 1998]. Hensikten med applikasjonene var å undersøke hvordan mobile feltarbeidere kunne bli støttet ved hjelp av kontekstrelevante tjenester. En av disse applikasjonene var Stickemap[1]. Denne ble testet ut av en økolog under et feltstudium i Kenya, hvor giraffenes matvaner ble observert over en to måneders periode. Utstyrt med en PalmPilot og en GPS-mottaker (se Figur 4), brukte økologen applikasjonen som et redskap for å registrere feltobservasjoner i kontekst.



Figur 4: Stickemap – Skjermdumpen til venstre viser brukerens posisjon i forhold til to observasjoner. Til høyre ser man økologen med en PalmPilot, og en giraff i bakgrunnen. (Bildene er hentet fra [Pascoe 1998]).

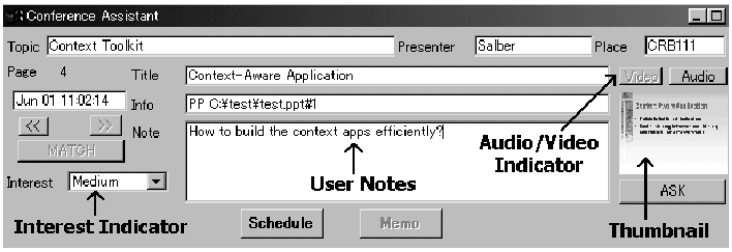
Stickemap benyttet kunnskap om brukerens posisjon til to formål; den viste økologen hvor på kartet hun befant seg, og knyttet automatisk observasjonene hun gjorde til stedene der de ble forfattet. Disse ble markert på kartet som små ikoner.

[1] Stickemap ble utviklet ved hjelp av Stick-e notes-arkitekturen (jamfør Kapittel 3.3, og 10.2).

4.3 The Conference Assistant

The Conference assistant er en prototyp på en mobilitetsbasert kontekstoppmerksom applikasjon [Dey et al. 1999a]. Dens formål er å være en slags digital assistent som kan holde rede på informasjon av interesse for en konferansedeltager. Ved hjelp av en rekke hjelpeservere stasjonert rundt i konferanselokalene, bevarer applikasjonen opplysninger om de forskjellige aktivitetene som skal avholdes. Dette inkluderer tidsrommet de spenner over, deres tema, og eventuelle forelesningspapirer (”slides”) som vil bli presentert. I tillegg bevarer applikasjonen opplysninger om alle deltagerne. Dette innbefatter deres interesser og preferanser, men også en oversikt over hvor de befinner seg i lokalene. Til sammen benyttes denne informasjon til å veilede deltageren rundt på konferansen, og informere om aktiviteter som pågår eller er i ferd med å begynne.

Et eksempel på hvordan The Conference assistant virker, illustreres når en deltager ankommer en forelesning. Hans oppmøte blir av applikasjonen automatisk registrert, og medfører blant annet til at informasjon om forelesningen hentes frem på PDA-en. Applikasjonen settes samtidig i en slags forelesningsmodus, noe som medfører at forelesningspapirene dukker opp som små ikoner på dataskjermen (se Figur 5). Applikasjonen sørger selv for å oppdatere denne informasjonen underveis, slik at det til enhver tid er samme papir på skjermen hos brukeren, som på lerretet hos foreleseren. Dette gjør det også enklere for brukeren å føre egne notater mens aktiviteten pågår: Disse notatene assosieres nemlig automatisk til papiret som fremvises, innad i applikasjonen.

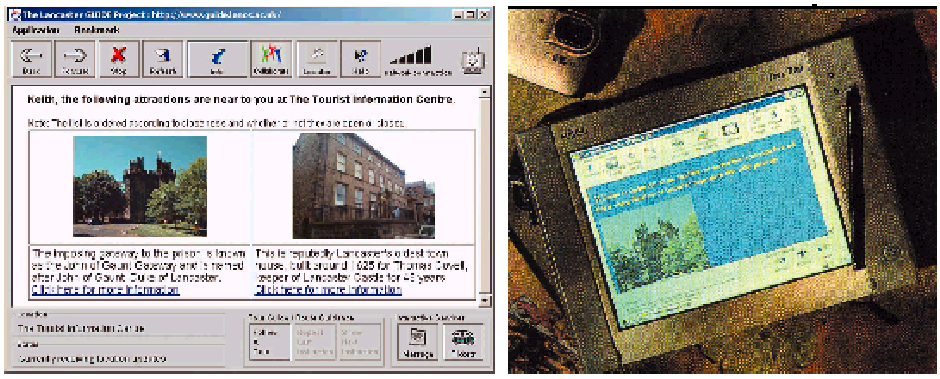


Figur 5: The Conference assistant – Slik ser applikasjonen ut under en forelesning. Til høyre vises det aktuelle papiret som blir presentert, mens man i midten kan man føre egne kommentarer. (Bildet er hentet fra [Dey et al. 1999a]).

Ettersom applikasjonen kjenner sin egen brukers interesser, kommer den stadig med forslag til hvilke aktiviteter han bør overvære under konferansen. Disse forslagene tar også hensyn til hvilke forelesninger hans kolleger skal delta på, siden dette vurderes som viktig sosial kontekst også for brukeren. Applikasjonen har i tillegg god kontroll på hvor kollegene befinner seg, og fremviser derfor deres lokasjon til brukeren om ønskelig.

4.4 The Lancaster Guide

Den siste eksterne applikasjonen jeg vil studere, er The Lancaster Guide [Davies et al. 2001, Cheverst et al. 2000]. Formålet med denne applikasjonen er å veilede turister rundt i byen Lancaster i England. Utstyrt med en håndholdt TeamPad (se Figur 6), har turistene mulighet til på egen hånd å forflytte seg rundt og få presentert informasjon om stedene de besøker. Applikasjonen benytter et trådløst nettverk til to formål: Registrering av brukerens lokasjon[1], og nedlasting av kontekstrelevant informasjon. For eksempel vil HTML-dokumenter som beskriver Lancasters slott dukke frem på PDA-en om slottet besøkes.



Figur 6: The Lancaster Guide – Bildet til venstre viser hvordan applikasjonen ser ut når to attraksjoner i nærheten av brukeren blir presentert. Til høyre vises en TeamPad hvor applikasjonen ligger installert. (Bildene er hentet fra [Davies et al. 2001]).

The Lancaster Guide tilbyr også skreddersydde turer rundt i Lancaster. En slik tur baseres på brukerens egne ønsker og preferanser, slik at om han oppgir at han har satt av tre timer og er interessert i historie, kan en tur innom galleriet og Lancasters slott typisk bli generert. I tillegg vil en detaljert beskrivelse over veivalg, foreslås. Likt som for resten av turen vil også denne bli generert ut i fra brukerens interesser. Formålet er å la han oppleve flest mulig severdigheter på turen mellom hovedseverdighetene. Dette algoritmen tar også hensyn til forhold som regnvær og lignende (brukerens fysiske kontekst), og forsøker å styre unna dette om mulig.

Applikasjonen gir også brukeren en oversikt over hvor andre i reiseføLget befinner seg. Disse personene kan videre utveksle kontekstrelevant informasjon mellom hverandre ved hjelp av et felles distribuert oppbevaringssystem: Notatene de forfatter for attraksjonene de besøker, blir automatisk gjort tilgjengelig for alle.

[1] Applikasjonene bestemmer brukerens lokasjon ut ifra nærmeste aksesspunkt til nettverket. Siden disse punktene (radiosenderne) er utplassert med cirka hundre meters avstand, blir også nøyaktigheten i forhold til brukerens lokasjon i samme størrelsesorden (+/- hundre meter).

4.5 Forbehold

Jeg har i dette kapitlet presentert fire eksterne eksempelapplikasjoner som alle var mobilitetsbaserte og kontekstoppmerksomme. Selv om jeg mener å ha fått et godt innblikk i hvordan disse applikasjonene virket, vil jeg likevel fremsette et forbehold. Siden jeg ikke har hatt en fullstendig tilgang på verken dokumentasjon eller kildekode for hver enkelt applikasjon, kan jeg ha også unngått å få med meg viktige detaljer vedrørende deres funksjonalitet. I noen tilfeller kan jeg også ha misforstått hvordan de faktisk har fungert. Dette vil i så fall kunne innvirke på resultatene jeg vil komme frem til i oppgaven.

Kapittel 5

Mocado

Etter å ha presentert fire eksterne eksempelapplikasjoner, har jeg nå kommet frem til mine to egne: Mocado og Mobitras. I dette kapitlet, og det neste, vil jeg presentere og kritisere arbeid og resultater gjort fra dette studiet. Selve analysen vil imidlertid presenteres samlet i Kapittel 7 og 8. Selv om Mocado er temaet for dette kapitlet, og Mobitras i neste, vil jeg likevel innlede med å si noen ord om begge.

Motivasjonen for å konstruere og implementere Mocado og Mobitras, var at jeg på et tidlig stadium i oppgaven ønsket å tilegne meg kunnskap om den typen applikasjoner Condor skulle støtte. Dette inkluderte hvilken funksjonalitet slike applikasjoner burde tilby, hvordan denne funksjonaliteten kunne bli konstruert, og ikke minst hvordan den var å anvende i praksis. Disse punktene påvirket hverandre under utviklingen, slik at bruken i praksis, ga inspirasjon og tilbakemelding om relevant funksjonalitet. Dette igjen, trakk meg tilbake til konstruksjonsbordet.

I utgangspunktet var det meningen å utvikle kun én applikasjon: Mocado. Mocado er en slags turistapplikasjon på lik linje med Cyberguide og The Lancaster Guide. Av den grunn erfarte jeg også etter hvert at jeg fikk relativt like applikasjoner å analysere. Om man skal være i stand til å skille de mer generelle kravene fra de spesielle, bør eksempelapplikasjonene variere noe mer [Roberts et al. 1996]. Av den grunn valgte jeg også å utvikle Mobitras. Mobitras skiller seg relativt mye fra de andre applikasjonene ved at den er tiltenkt idrettsutøvere og mosjonister.

Selv om Mocado og Mobitras henvender seg til to vidt forskjellige brukergrupper, er de likevel identiske med tanke på utstyr og teknologi. Applikasjonene er utviklet i Java standard edition på en vanlig PC, og testet ut på noen ulike versjoner av Cassiopeia Pocket-Pc (se Figur 1). To forskjellige kjøreomgivelser (Java Runtime Environments) er også utprøvd. Den eldste, en betaversjon av PersonalJava RE 1.0, var etter mine erfaringer den mest stabile (den andre versjonen var PersonalJava RE 1.1). Begge versjonene er fra Sun og beregnet for operativsys-

temet WinCE (som også Cassiopeiaen benyttet). Da begge applikasjonene var utviklet for utendørs bruk, ble en GPS-mottaker benyttet for registrering av posisjon (se Figur 7).

Figur 7: Mocado-, og Mobitrasutstyr – Bildet viser utstyret som ble anvendt under uttesting. Øverst ses en Garmin GPS-mottaker tilkoblet en Cassiopeia PPC med et digitalt kamera festet på toppen.

5.1 Innledende om Mocado

> *”Etter å ha tilbrakt dagen i Vigelandsparken og på Akershus festning, har Kari funnet veien hjem til hotellrommet. Med PDA-en i hånden, sitter hun tilbakelent i stolen for å kikke igjennom dagens utflukt. Fordi Mocado på egen hånd har gjort opptak av hennes posisjonering i løpet av dagen, blir den tilbakelagte ruten tegnet opp som en rød snor på kartet.*
>
> *Som et resultat av Karis mange forfattede notater, ligger det nå en rekke små ikoner spredd utover langs ruten på kartet. Kari trykker på de forskjellige, og får frem bildene fra turen sammen med kommentarene som hører til. ”*

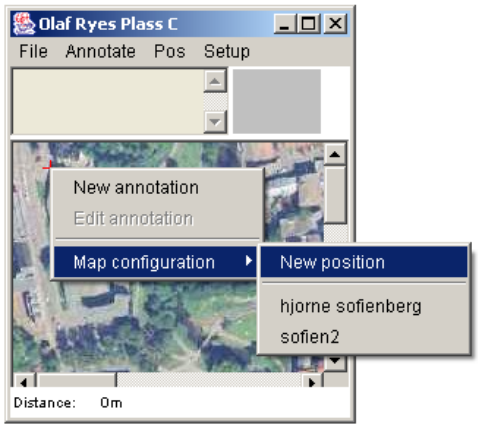
Utgangspunktet for Mocado (”Mobile Context-aware Annotation Demonstrator”) var å lage en kontekstoppmerksom elektronisk guide som kunne bistå ferierende turister. Jeg ønsket at turisten, ved hjelp av sin egen PDA og GPS-mottaker, kunne få hjelp til å finne frem i ukjente byer eller tettsteder. I tillegg skulle han få informasjon om severdigheter i nærheten av der han var. En slik orientering kunne for eksempel være opplysninger om en statue han observerte, eller når neste buss var ventet på holdeplassen ved siden av.

Underveis i utviklingen identifiserte jeg relativt mye funksjonalitet jeg vurderte som relevant. En del av dette ble også implementert. I dette kapitlet har jeg valgt å drøfte både hvordan Mocado faktisk virker, samt hvordan jeg i ettertid mener den burde ha virket. Jeg vil imidlertid gjøre det klart hva som faktisk er realisert, og hva som er tanker og ideer. Applikasjonen i seg selv ble utviklet over en fem måneders periode (fra august 2000, til januar 2001), og anvender 53 Java-klasser med til sammen 5143 linjer kildekode (se også vedlegg).

5.2 Kart og posisjon

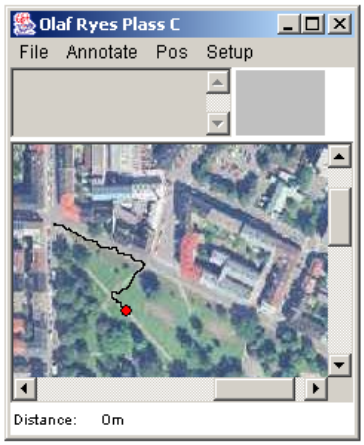
En av de mest sentrale tjenestene Mocado tilbyr, er å vise turisten hvor han befinner seg på et kart. Ettersom Mocado bare er en prototyp, har jeg valgt å foreta noen forenklinger. Dette inkluderer både hvordan jeg skaffer til veie kart, samt hvordan disse blir konfigurert. Denne avgjørelsen er gjort siden formålet med applikasjonen var å se hvilke kontekstoppmerksomme tjenester man kunne ønske seg tilgjengelig, fremfor hvordan virkelige GIS (Geographical Information Systems) fungerer i detalj. Kartene som benyttes er derfor bilder av typen GIF eller JPEG, og legges i en egen mappe på PDA-en før eksekvering.

Før kartbildene kan anvendes som kart, må de konfigureres. Dette uføres på følgende måte: Man begynner med å ta med seg alt utstyret ut (se Figur 7) og plassere seg på et velkjent sted så langt opp i hjørnet av kartet som mulig. Man markerer på kartbildet hvor man er (i Mocado), og lar applikasjonen motta en observasjon fra GPS-mottakeren (se Figur 8). Mocado blir dermed i stand til å assosiere pikselkoordinatene fra markeringen, med posisjonene fra mottakeren. Brukeren finner deretter et annet sted på kartet, helst så langt unna det forrige punktet som mulig diagonalt, og gjentar prosessen der. Målet er å oppnå en lengst mulig avstand mellom punktene, både horisontalt og vertikalt. Da vil nøyaktigheten til kartets skala bli best. Etter at disse punktene er blitt registrert, har Mocado nok informasjon til å regne ut både origo og skala. Denne informasjonen lagres på PDA-en, slik at konfigureringen ikke trenger gjentas. Imidlertid kan brukeren konstruere flere kart om ønskelig.



Figur 8: Konfigurering av kart – Skjermdumpen viser hvordan man kan konfigurere et kart ved hjelp av flere indekspunkter. Det lille trådkorset øverst i den indre dialogboksen viser hvor turisten trykket på kartet.

Når kartet er konfigurert, er applikasjonen klar for anvendelse. Ved hjelp av posisjonsdata fra GPS-mottakeren, registrerer Mocado hvor turisten befinner seg og tegner opp denne posisjonen som et rødt punkt på kartet (se Figur 9). Etter hvert som brukeren forflytter seg, blir en posisjonshistorie øyensynliggjort som en rød tråd.



Figur 9: Mocado og posisjonering – Her ser man brukerens posisjon sammen med en hale som indikerer hvor han kommer gående fra (halen på dette bildet er retusjert for å skape kontraster).

5.2.1 Tanker og erfaringer om kart og lokasjon

Erfaringene jeg gjorde når det gjaldt behandling av kart og lokasjon er relativt tilfredsstillende. Bruk av vanlige kartbilder fungerte som forventet sett i forhold til målet med applikasjonen. Konfigureringen var også rask og enkel, og tillot bruk av mange ulike typer bildekart. Flyfotoene (ortofotoene) jeg anvendte var ett eksempel, men også tegnede kart ble utprøvd. I mange tilfeller var disse

klarere på skjermen til Cassiopeiaen, spesielt i dagslys. Da ble flyfotoene noe ugreie å se på fordi PDA-ens lysstyrke var litt svak. I tillegg fikk tegnede kart plass til mer informasjon av typen gatenavn og lignende. Flyfotoene, derimot, kunne avsløre stier, benker, og trær. Dette gjorde flyfotoene både morsommere og mer nøyaktige å jobbe med. Spesielt det siste var av interesse under uttestingen.

GPS-sensoren jeg benyttet hadde stort sett en nøyaktighet på rundt +/- 5 meter[1]. Noe fluktuering forekom iblant (vandring rundt brukerens egentlige posisjon). Ettersom posisjonene som ble mottatt var ustabile, og viste ulike verdier selv når jeg sto helt stille, prøvde jeg en alternativ metode under kartkonfigureringen: Ved å ta gjennomsnittet av mer enn én posisjon for hvert av de to referansepunktene, kunne jeg foreta en tilnærming til brukerens faktiske posisjon på hvert sted. Jeg forsøkte meg frem med noen ulike mengder, og fant ut at femten posisjoner for hvert punkt var tilstrekkelig. Da ble opptegningen av brukerens posisjon vesentlig mer nøyaktig.

Over enkelte geografiske områder benyttet jeg også kart med ulik skala, eller kart som overlappet hverandre. En funksjon som kunne ha vært implementert, men som ble unnlatt grunnet tidsrammene, ville ha vært å la applikasjonen hente frem og skifte mellom disse automatisk i forhold til brukerens posisjon. Slik Mocado er i dag, må dette gjøres manuelt. På sikt kunne jeg også tenkt meg å implementere deler av kartfunksjonaliteten ved hjelp av trådløs tilgang til en GIS-database.

5.3 Opptak og avspilling

Som en tilleggsfunksjon til kart og posisjon, kan turisten gjøre opptak av sine egne turer (se Figur 10). Ved å lagre slike turer, kan han også spille de av på et senere tidspunkt. Applikasjonen vil da oppføre seg som om brukeren gikk turen på nytt, men ikke nødvendigvis slik den gjorde da opptaket opprinnelig ble gjort. Dette kommer av at nye konfigureringer og annoteringer vil kunne endre applikasjonen over tid (jeg vil beskrive annoteringer nedenfor). Ettersom Mocado ikke skiller mellom om man er ute og går, eller bare spiller av et gammelt opptak, har man også tilgang til mye av den samme funksjonaliteten som ellers[2]. Dette gjør at man både kan konfigurere kart, eller konstruere annoteringer, under avspillingen. Om ønskelig har man også mulighet for å velge et annet avspillingstempo enn det som opptaket opprinnelig ble gjort i.

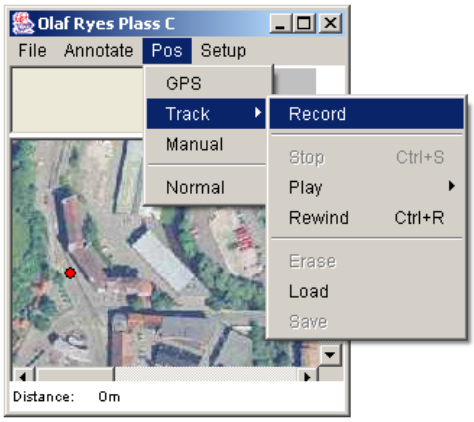
---

[1] Nøyaktigheten til GPS-mottakerne har en tendens til å variere litt i forhold til hvor GPS-satellittene befinner seg over jorden. Disse beveger seg i geosynkrone baner, og dekker dermed ulike områder med varierende kvalitet til forskjellige tider.

[2] Fordi man allerede bruker opptak-, og avspillingsfunksjonaliteten til å spille av en tur, har man ikke mulighet til å gjøre et nytt opptak samtidig.

5.3.1 Tanker og erfaringer om opptak og avspilling

Selv om jeg i ettertid oppfatter opptak og avspilling som en interessant og nyttig egenskap (en slags historielogging), var det ikke opprinnelig det som var hensikten. Formålet var å redusere antall turer man som utvikler må ta frem og tilbake mellom utviklingsmaskin og testområde under implementasjonen. Fremgangsmåten er gjerne slik at man tester applikasjonen ute, finner en (eller flere) feil, går inn og endrer litt, kompilerer og overfører dette til PDA-en, går ut og tester, finner nye feil som har kommet til, går inn igjen og retter, og så videre. Ofte kan små feil i kildekoden resultere i utallige turer ut og inn, noe som i praksis tar veldig mye tid. Med mulighet for opptak og avspilling ble tiden benyttet til uttesting redusert betraktelig, og mer ressurser kunne avsettes for identifisering og bygging av ny funksjonalitet.



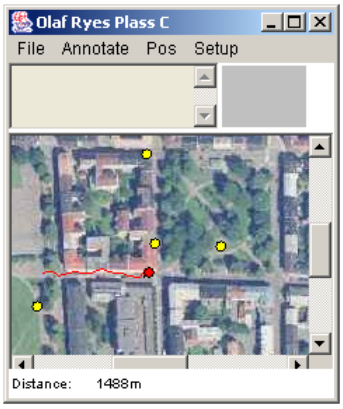
Figur 10: Opptak og avspilling – Skjermdumpen viser hvordan brukeren kan foreta opptak og avspilling av turer som tilbakelegges.

I ettertid ser jeg nytteverdien av kontekstopptak også for kontekstoppmerksomme tjenester. Dette inkluderer spesielt bruk av konteksthistorie. Hvilke situasjoner brukeren har opplevd i fortid, kan ofte være med på å forklare hans situasjon i nåtid. Slik sett kunne det vært spennende å utnyttet dette aktivt. For eksempel kunne slike opplysninger blitt anvendt for å presentere relevant informasjon i forhold til brukerens situasjon. Slik Mocado er i dag, benyttes bare hans lokasjon i nåtid for å avgjøre dette. Som jeg vil komme tilbake til nedenfor, var det ofte et behov for å variere informasjonen som ble presentert ved andre eller tredje aktivering. En aktiveringshistorikk kunne ha muliggjort dette.

Om Mocado skulle oppbevart en logg over brukerens kontekst, vil jeg hevde at hans spor måtte bli ivaretatt mer kontinuerlig. Aller helst burde dette systemet atskilles fra hans egne personlige turopptak. Dette ville selvfølgelig stilt høyere krav til både lagringskapasitet og prosessorkraft, noe som fortsatt er relativt beskjedne størrelser hos PDA-ene. For eksempel var Cassiopeiaens minnekapasitet på 32MB. Et fem minutters turopptak tok cirka 10KB, slik at om 32MB var avsatt for en slik historikk, vil minnekapasiteten vart i overkant av tre uker.

5.4 *Annoteringer*

Selv om bruk av kart og lokasjon kan vurderes som sentralt i Mocado, er det likevel alt man kan bygge på bakgrunn av dette som gir grunnlag for de mest interessante kontekstoppmerksomme tjenestene. I denne oppgaven er tjenesten jeg har rettet søkelyset på, anvendelse av kontekstrelevant informasjon. I Mocado har jeg valgt å benytte annoteringer for å demonstrere dette konseptet. En annotering er en slags elektronisk post-it lapp[1] som, fremfor å bli limt fast på et eller annet objekt, assosieres med en eller annen lokasjon. Eksempelet i begynnelsen av kapitlet illustrerte slike "lapper"[2]. Alle bildene Kari tok, og kommentarene hun skrev, ble automatisk "festet" til stedene der hun var. I tillegg ble de tegnet opp på kartet som små gule punkter (se Figur 11). Hensikten med annoteringene er å gi turistene anledning til å forfatte og motta relevant informasjon i henhold til deres posisjon.



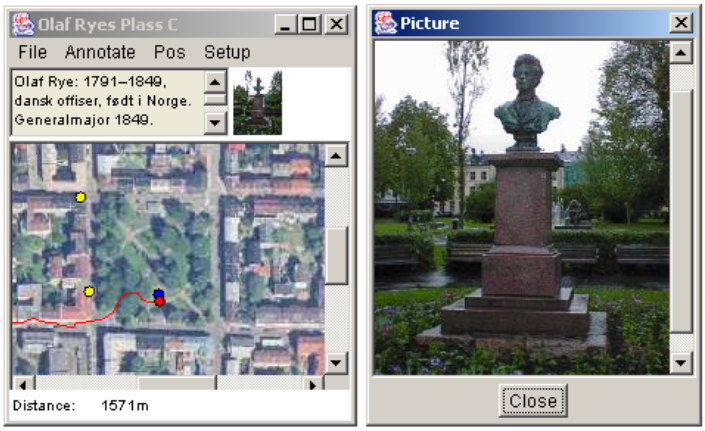
Figur 11: Mocado med annoteringer – Her kan man se turisten omgitt av flere annoteringer.

En annotering i Mocado kan bestå av tre typer informasjon; en tekst, et bilde, og et lydklipp. Teksten kan føres inn direkte, mens både bilde og lyd må fanges av andre applikasjoner først, for så å legges inn i applikasjonen senere. Hver annotering assosieres videre med en lokasjon og en radius som til sammen beskriver området hvor informasjonen skal gjelde. Radiusen kan gjerne variere fra annotering til annotering, og fastsettes av turisten når informasjonen forfattes. En annotering for en statue kan for eksempel ha en radius lik fem meter, mens en annotering for et fotballstadion kan ha en kilometer. Straks brukeren beveger seg innenfor annoteringens areal aktiveres den, og dens informasjon dukker opp på PDA-en (informasjonen blir gyldig). I tillegg skifter annoteringen farge fra gult til blått på kartet. Om ønskelig kan også annoteringens lydklipp spilles av når aktiveringen inntreffer (forutsatt at annoteringen har et slikt assosiert).

---

[1] Begrepet "elektroniske post-it lapper" er hentet fra Browns Stick-e notes [Brown 1995].

[2] I eksemplet kalte jeg annoteringene for notater.

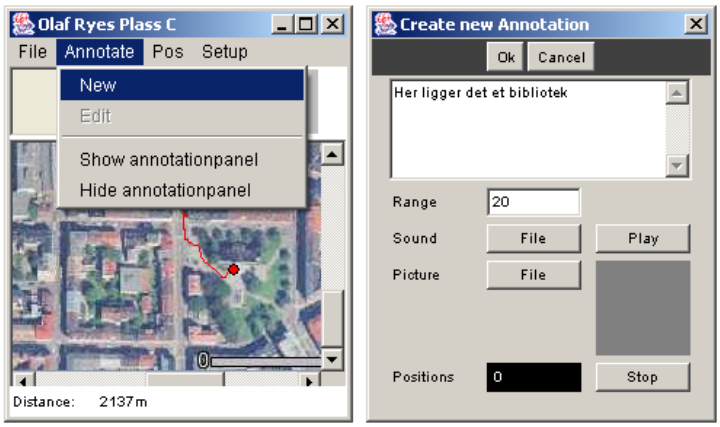


Figur 12: Mocado under aktivering – Her ser man brukeren som har aktivert en annotering festet på statuen av Olaf Rye. Bildet til høyre dukker opp om man trykker på den lille versjonen til venstre.

I et vanlig brukstilfelle med Mocado, vil applikasjonen presentere relevant informasjon om en serverdighet brukeren besøker. På samme måte vil informasjonen forsvinne igjen når besøket er over. Dette er hva jeg tidligere har definert som aktivering og deaktivering av kontekstrelevant informasjon. Begge deler kan skrus av og på i menyen. Det siste er hensiktsmessig da det ikke bestandig er sikkert at man vil ha slik informasjon presentert automatisk. Noen ganger er det mer behagelig å kontrollere dette selv. For eksempel kan man ønske å få frem informasjon om en severdighet man ser lengre unna, uten at nære annoteringer aktiveres og dekker over denne informasjonen. Det samme prinsippet gjelder også deaktiveringen; selv om brukeren beveger seg vekk fra en severdighet, kan det likevel hende at han leser eller hører på dens informasjon. I så fall burde han selv kunne bestemme når annoteringen skal lukkes (deaktiveres).

En annotering kan konstrueres på to måter. Enten ved å gå direkte i menyen og velge ”ny annotering” (se Figur 13), eller ved å markere på kartet der man ønsker at den skal ligge (se Figur 8). Ved å velge det første, blir annoteringen automatisk festet til stedet der den forfattes. For å fastsette annoteringens posisjon, anvendes samme metode som under kartkonfigureringen; mens brukeren forfatter annoteringen, tar applikasjonen i mot GPS-posisjoner. Man bør derfor holde seg i ro på stedet til femten posisjoner er mottatt, noe som i praksis tar et halvt minutt. Om man heller velger den andre metoden (å markere på kartet hvor annoteringen skal ligge) har ikke applikasjonen behov for å kjenne brukerens posisjon. En slik annotering kan med andre ord også skapes innendørs, selv uten at GPS-mottakeren er koblet til (GPS fungerer som nevnt ikke innendørs).

For å redigere eller fjerne en annotering, anvendes samme metode som over; enten ved å velge ”rediger” i menyen, eller ved å markere den ønskede annoteringen direkte på kartet. Det første alternativet er til for å redigere annoteringen som presenteres idet valget utføres.



Figur 13: Konstruksjon av annoteringer – Skjermdumpen til venstre viser hvordan annoteringer legges inn ved hjelp av menyen. Til høyre vises dialogvinduet som dukker frem. Slike annoteringer assosieres direkte til turistens posisjon.

5.4.1 Tanker og erfaringer om annoteringene

En målsetting jeg hadde da jeg gikk i gang med Mocado, var at applikasjonen skulle være brukervennelig. Spesielt gjaldt dette annoteringene; det skulle være enkelt å både forfatte og publisere slike. Jeg var av den oppfatning at om applikasjonen skulle appellere til turister, burde hele prosessen med å forfatte nye annoteringer gå raskt. Hvor raskt, var selvsagt betinget av mengden informasjon som ble ført inn. Aller helst ønsket jeg at dette kunne bli utført med en hånd, omtrent som å bruke en diktafon. Tanken var at turisten ikke nødvendigvis måtte stoppe opp for å rette all oppmerksomhet på PDA-en under konstruksjonen av en ny annotering.

I praksis lykkes jeg kun delvis med kravet ovenfor. Som nevnt ble både innspilling av lyd og bildefotografering implementert slik at man var tvunget til å anvende andre eksterne applikasjoner på PDA-en. Forfatting av tekst stilte også naturlige krav til at brukeren stoppet litt opp; det er vanskelig å gå og skrive samtidig. Om annoteringen ble plassert der brukeren sto, hadde applikasjonen også behov for å motta femten observasjoner fra GPS-mottakeren (for å beregne hans nøyaktige posisjon). Et halvt minutt måtte derfor avsettes til dette. Som oftest var det likevel forfattingen av informasjon som tok lengst tid. Det er slitsomt å skrive mye tekst med skriftgjenkjenning på en PDA. Begrensingene i applikasjonen (med tanke på rask forfatting) føltes derfor ikke så hemmende i forhold til begrensningene i utstyret (PDA og GPS-mottaker).

Det dukket opp minst to problemstillinger når det gjaldt aktivering og deaktivering av annoteringer. Det første var at annoteringer gjerne kan overlappe eller dekke hverandre helt. For eksempel kan en fornøyelsespark ha én stor annotering med generell informasjon festet til hele parkområdet i tillegg til mange små rundt hver karusell. Hva slags regler skal da gjelde for aktivering og deaktivering av disse? Hvordan man velger å løse dette avhenger mye av hvordan man ønsker at

applikasjonen skal fungere. I Mocado bestemte jeg meg for at følgende skulle gjelde: Hvis brukeren befant seg innenfor en annotering og beveget seg over i en annen som overlappet, skulle Mocado deaktivere den første (den som blir vist), og aktivere den nye. Når brukeren senere beveger seg ut av denne igjen, og tilbake til den som ble overlappet (og som brukeren derfor har aktivert tidligere), skal den nye aktiveres på nytt mens den andre deaktiveres. I eksempelet med fornøyelsesparken vil dette bety at når man kommer inn i parken, så aktiveres selve parkannoteringen. Når man deretter kommer i nærheten av en karusell, lukkes parkannoteringen, og karusellannoteringen aktiveres. Beveger man seg bort fra karusellen igjen, aktiveres parkannoteringen på nytt. Selv om dette systemet fungerte i de fleste situasjoner, hadde det likevel en sideeffekt; om annoteringene spilte av et lydklipp, eller på en annen måte "forstyrret" brukeren, kunne dette oppleves som meget slitsomt. Om for eksempel parkannoteringen spilte av en velkomsttale hver gang den ble aktivert, ville dette inntreffe hver gang man beveget seg vekk fra en karusell. Jeg kunne derfor ha tenkt meg en metode for å spesifisere flere versjoner av en og samme annotering: En som ble presentert ved første aktivering, og en som ble presentert ved andre.

En annen observasjon jeg gjorde, var at jeg enkelte ganger "mistet" relevant informasjon. Om flere overlappende annoteringer var aktivert etter hverandre, ville man miste de som var aktivert først. I parkeksempelet ville for eksempel parkannoteringens informasjon forsvinne når en attraksjonsannotering ble aktivert. På en måte er dette litt uheldig, ettersom all aktivert informasjon er relevant informasjon, og all relevant informasjon skal presenteres. I noen tilfeller kunne dette dreie seg om både tre og fire overlappende annoteringer samtidig. Et eksempel var når jeg sto ved siden av en statue i Vigelandsparken. Da hadde jeg aktivert én stor annotering for hele Oslo, én litt mindre for parken, og tilslutt en liten annotering for selve statuen. Kun den siste ble fremvist. En oversikt over alle aktiverte annoteringer ville ha vært informativt, gjerne kombinert med muligheten for å bytte mellom hvilken som ble presentert.

Til sist registrerte jeg et problem når brukeren beveget seg langs grensen til en annotering. Dette kunne medføre at annoteringen ble aktivert og deaktivert flere ganger etter hverandre. Dette oppsto gjerne som en følge av at brukeren beveget seg like innenfor og utenfor grensen til annoteringen[1]. På grunn av tidsrammene for Mocado fikk jeg ikke lagt inn noen egen funksjonalitet for å bøte på dette. Jeg tok isteden med meg disse erfaringene videre til utviklingen av Mobitras. Jeg vil beskrive hvordan jeg valgte å løse dette problemet der, i neste kapittel.

Selv om en annoterings gyldighetsområde blir bestemt av dens posisjon og radius, ønsker man som oftest likevel å knytte dem til et mer bestemt areal. I Vigelandsparken kunne jeg for eksempel tenkt meg at parkannoteringen dekket hele parken, og bare den. Slik det er nå, blir dette arealet isteden definert av én stor sirkel.

[1] Dette ble også forsterket om man opplevde mye fluktuering i posisjonene fra GPS-mottakeren.

5.5 Manuell modus

I mange tilfeller kan det være praktisk å aktivere annoteringer som befinner seg på andre steder enn der man er. På samme vis kan det også være behagelig å simulere at man befinner seg et annet sted og i en helt annen situasjon. I Mocado har jeg valgt å kalle dette for *manuell modus*. Når applikasjonen settes i denne modusen vil den tolke enhver markering på kartet som om brukeren faktisk flyttet seg dit. Dette medfører at det røde punktet som viser hvor man er på kartet, automatisk flyttes etter. Applikasjonen skiller heller ikke mellom hvorvidt posisjonsangivelsene kommer fra GPS-mottakeren eller brukerens markeringer. Dette gjør at man har tilgang til den samme funksjonaliteten som ellers. Spesielt vil en forflytting innenfor radiusen til en annotering medføre at annoteringen aktiveres. Mocado inneholder med dette fire forskjellige modi: Normal, GPS, opptak og avspilling (som jeg vurderer som ett modus), og manuell modus. Hvorfra applikasjonen mottar nye posisjonsangivelser avhenger av disse, og bestemmes av brukeren i menyen (se også Figur 10):

Normal	Dette er modusen Mocado er satt i når den startes opp. Her får den ikke posisjonsangivelse noestedsfra.
GPS	Her mottas all posisjonering fra GPS-mottakeren.
Opptak og avspilling	Her mottas all posisjonering fra avspillingsfunksjonen.
Manuell	Her bestemmer brukeren selv hvor han skal plasseres på kartet.

5.5.1 Tanker og erfaringer om manuell modus

Jeg synes muligheten for å kunne plassere seg selv i tenkte situasjoner er en spennende funksjonalitet: Det å kunne aktivere informasjon fra andre steder og tider enn der man selv befinner seg. Brown hevder at man mer generelt bør ha mulighet til følgende:

> *"...the user should be allowed to pretend that they were in a location other than their real one: carrying this to an extreme, the user may be a surrogate tourist sitting in an armchair, and simulating a walk round a city (e.g. by pointing at a map), triggering information as he goes."*
>
> – [Brown 1998a, side 3].

Det å kunne sitte hjemme i sin egen lenestol mens man besøker Trafalgar square, er en forlokkende tanke. Dette blir imidlertid en liten fremtidsvisjon slik Mocado fungerer i dag. Dette skyldes mengden informasjon som er lagt inn: Antall annoteringer begrenser seg kun til de jeg har benyttet under uttestingen. Det samme gjelder kartverkene. Dessuten er lokasjon den eneste parameteren som blir benyttet for å beskrive brukerens kontekst. Likevel er det fristende å tenke litt fremover. Hva om man får et system hvor "alle" kan forfatte og publisere annoteringer på samme måte som html-dokumenter utveksles på Internett i dag? Hva om kontekst og kontekstrelevant informasjon oppbevares i en historielogg? Kanskje kan jeg benytte situasjonssimulering en gang i 2025 til å se hvordan en spasertur langs Akerselven ville fortonet seg i år 2003.

Litt omvendt i forhold til erfaringene gjort fra opptaksfunksjonaliteten, fant jeg at manuell modus også var en nyttig egenskap under applikasjonsutviklingen. Nå var jeg ikke engang nødt til å gjøre opptakene ute før jeg testet applikasjonen; jeg kunne simulere spaserturene inne. Dette gjorde det vesentlig enklere å avgjøre hvordan applikasjonen oppførte seg i sjeldent inntrufne situasjoner. For eksempel kunne jeg simulere hvordan Mocado fortonet seg for en turist som beveget seg langs kanten av en annotering. Det er imidlertid et faretegn om alle prøveturene blir gjennomført på utviklingsmaskinen inne. Noe av hensikten med prosjektet, som jeg også nevnte innledningsvis, var å se hvordan applikasjon fungerte i praksis. Dette innebar blant annet hvordan det føltes å spasere med utstyret i hånden, bruk av applikasjonen under forskjellige værforhold, folk som snur seg på gaten og lurer på hva du holder på med, og lignende. Dette er forhold som er viktig for helhetsopplevelsen, men som ikke oppdages om man ikke prøver applikasjonen selv. Det var i disse situasjonene jeg så nye fordeler og ulemper ved bruk av ulike karttyper (flyfoto eller tegnede kart). Hvordan trafikkstøy kunne gjøre det umulig å høre lyden som ble spilt av på PDA-en, og maskinens treghet når det ble vinter og kaldt. Det siste, lav temperatur, gjorde at Mocado enkelte ganger eksekverte så tregt at den hang seg opp. Dette kunne oppleves som meget ergerlig om nye annoteringer var lagt inn, men ennå ikke lagret[1]. Dette resulterte i en endret versjon hvor annoteringer (og kartkonfigureringer) ble lagret umiddelbart etter konstruksjon. Under varmere forhold fungerte imidlertid maskinen som forventet.

5.6 Oppsummering

Mocado er en prototyp på en mobilitetsbasert kontekstoppmerksom applikasjon. Dens hovedfunksjonalitet er å vise brukeren hvor han befinner seg på et kart, og tilby kontekstrelevant informasjon i form av annoteringer. Grunnen til at jeg valgte å utvikle nettopp Mocado, var for å oppnå førstehåndskunnskap om appli-

[1] Cassiopeiaen hadde egentlig ingen harddisk, men en felles minnekapasitet på 32MB som dynamisk ble fordelt mellom programvare og kjøreomgivelser.

kasjonstypene rammeverket jeg skal frem til skal støtte. Jeg ønsket å undersøke hva slags funksjonalitet man har behov for i slike applikasjoner, og hvordan denne funksjonaliteten kan anvendes i praksis. Jeg vil trekke frem fire viktige erfaringer jeg gjorde rundt dette:

Bruk av sensorer – Første milepæl i utviklingen var å anvende kart og lokasjon. I begynnelsen var dette langt fra enkelt, spesielt når det angikk hvordan man skulle gå frem for å opprette kommunikasjon med GPS-mottakeren. En ekstra utfordring var også å få dette til å virke i kjøreomgivelsene til Cassiopeiaen (som benyttet en mindre versjon av API-et som fulgte med JDK 1.3). Deretter var jeg avhengig av å "oversette" dataene fra mottakeren over til et mer applikasjonsvennelig format (fra UTM til pikselkoordinater). Dette er også i samsvar med erfaringene andre har gjort; Dey, med flere, hevder at en av hovedgrunnene til at kontekstoppmerksomhet blir så lite anvendt, skyldes problemene som ligger i å ta i bruk og endre formatet på dataene som mottas fra sensorene [Dey 2000, Pascoe 1998]. Når dette var på plass, følte jeg at mange av usikkerhetsmomentene med applikasjonen ble borte, og jeg kunne sette i gang med å konstruere og implementere de mer kontekstrelevante tjenestene.

Aktiveringer – Jeg identifiserte to behov vedrørende aktivering av annoteringer. Det første behovet dreide seg om det var hensiktsmessig å presentere forskjellig informasjon ved første og annen aktivering. Det andre gjaldt problematikken når en bruker beveget seg langs kanten av en annotering. Når det gjaldt det første, erfarte jeg at det var et helt klart ønske om å kunne skille mellom aktiveringene. Dette var spesielt fremtredene om annoteringene overlappet hverandre. Ofte ønsket jeg å presentere forskjellig informasjon både mellom første, andre, og tredje aktivering. Det andre problemet, grenseproblematikken, viste seg relativt ofte om forholdene til GPS-mottakeren var dårlige. Selv om jeg unnlot å rette på disse kravene i denne prototypen, mener jeg at de bør behandles i en eventuell kommersiell versjon.

Overlappende annoteringer – Det var langt fra uvanlig at annoteringene jeg la inn overlappet. Jeg var derfor nødt til å implementere funksjonalitet som behandlet disse tilfellene på en logisk måte. Dette problemet mener jeg også kan generaliseres til å gjelde flere former for kontekstrelevant informasjon. Her var det annoteringsarealene som overlappet, i en annen applikasjon kan det være andre parametere og intervaller. I The Conference Guide [Dey et al. 1999a] kunne for eksempel tidsintervallene for de ulike forelesningene overlappe hverandre.

Manuell modus – Til sist så jeg også et klart behov for å kunne sette applikasjonen i en manuell modus. Selv om hovedfunksjonaliteten til Mocado var å presentere informasjon relatert til der man var, fant jeg det like fornuftig å kunne lese eller forfatte annoteringer også for andre områder. Dette kunne dreie seg om et bygg man så lengre unna, eller kanskje en begivenhet man kjente til et helt annet sted. Jeg mener derfor at det i Mocado var like viktig for en turist å ha tilgang til informasjon som befant seg utenfor hans personlige kontekst, som den som befant seg innenfor.

Som nevnt ønsket jeg å finne ut av hvordan applikasjonen var å benytte i praksis. Hva slags funksjonalitet kunne man for eksempel ha tenkt seg om man sto midt i en ukjent by med PDA-en i hånda? Er denne applikasjonen noe en turist faktisk har behov for, eller er alt utstyret bare forstyrrende for helhetsopplevelsen? Oppsummert har jeg for det meste gode erfaringer rundt dette. Om man ser bort fra at man må ha med seg en del utstyr (en Cassiopeia, to meter ledning, og en GPS-mottaker), virker applikasjonen i stor grad slik jeg så for meg under planleggingen. Det er enkelt å legge til nye annoteringer (både når man sitter hjemme i lenestolen eller er ute i parken), og den virker også tilfredstillende med tanke på hvordan annoteringene aktiveres ved bruk. Likevel er det vanskelig for meg å avgjøre applikasjonens informative nytte. Fordi det var jeg som både konstruerte den, la inn annoteringene, og til sist også testet den ut, er det vanskelig å bestemme om applikasjonen er like interessant også for turister. Cheverst hevder at hvis man virkelig skal evaluere en applikasjons brukbarhet, så må den prøves ut av de som den i utgangspunktet er tiltenkt for [Cheverst et al. 2000]. Jeg føler likevel at jeg har klart å fange noe av den funksjonaliteten man kan forvente fra slike applikasjoner (jeg har tross alt vært turist en og annen gang selv), og samtidig fått et godt inntrykk av hvordan slike applikasjoner kan være å anvende i praksis.

Til sist vil jeg nevne at selv om Mocado i utgangspunktet var tiltenkt turisme, har jeg stor tro på at den også kan benyttes innenfor andre områder. En rørlegger som jobber ute, for eksempel, kan bruke den til å navigere rundt fra sted til sted under arbeidsdagen. Annoteringsfunksjonaliteten kan hjelpe til med å finne frem til rør som ligger gjemt under bakken, og videre benyttes til å spesifisere faglige notater og erfaringer på de samme stedene. Om denne informasjonen i tillegg kunne bli lest og redigert av alle kollegene i fellesskap, hadde man hatt et system for å bevare og formidle lokasjonsrelevant kunnskap over tid.

Kapittel 6

Mobitras

Etter at jeg så meg ferdige med første versjon av Mocado, vurderte jeg om jeg skulle fortsette utviklingen av denne ytterligere. Dette ville i så fall betydd eksperimentering med bruk av trådløs kommunikasjon mellom flere Mocado-brukere. Slik funksjonalitet ville ha muliggjort utveksling av annoteringer og posisjon mellom flere turister, og bidratt til å utvide bruken av kontekst til å innbefatte "nære personer" (sosial kontekst). Etter å ha vurdert mulighetene for videre utvikling av Mocado, lot jeg dette være. Hovedgrunnen var at det allerede forelå en del eksterne applikasjoner som lignet mye på Mocado (blant annet The Lancaster Guide og Cyberguide, se Kapittel 4), hvorav enkelte av disse allerede hadde implementert slik kommunikasjon. Jeg fant derfor ut at jeg hadde mer lyst til å lage noe helt annet, en applikasjon som ikke var en eller annen form for elektronisk guide. Resultatet ble Mobitras.

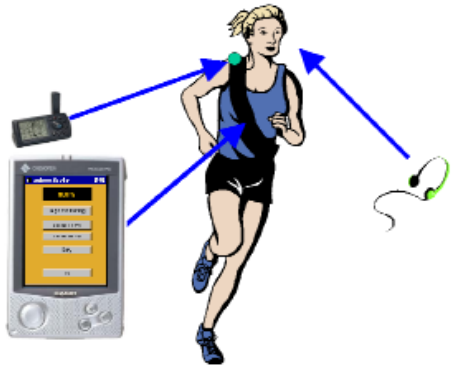
> *"Anne og Mobitras er i ferd med å tilbakelegge første runde rundt Sognsvann. Mobitras er et dataprogram installert på Annes PDA, og tjenestegjør som hennes personlige - assistent når hun er ute og trener. Etter hvert som Anne passerer de innlagte sekunderingspunktene rundt vannet, forteller Mobitras hvordan turen har forløpt seg så langt: Distanse, gjennomsnittshastighet, og antall forbrente kalorier er blant verdiene som leses opp."*

Hensikten med Mobitras (Mobile Training Assistant) er å tilby en elektronisk assistent for idrettsutøvere og mosjonister. Applikasjonen er i utgangspunktet tiltenkt langdistanseløpere, men skal også egne seg godt for andre lignende idrettsgrener som sykling og langrenn. Hovedsaken er at treningene består av å tilbakelegge lengre strekninger utendørs, og at dette blir gjort med noenlunde fri sikt mot himmelen (av hensyn til GPS-mottakeren). Målet er at utøveren, ved hjelp

av hodetelefoner og muntlige tilbakemeldinger, skal få tilgang til sekunderingsinformasjon om hvordan han ligger an underveis (se Figur 14). Dette er først og fremst opplysninger som kan fremskaffes ved hjelp av kunnskap om hans posisjonshistorie kombinert med tid.

> *"Idet Anne passerer benkene langs nordre del av vannet, hører hun plutselig at noen kommer løpende bak. 'Odin nærmer seg bakfra…' forteller Mobitras. Odin er ingen virkelig person, men skyggen av et løp Anne gjennomførte uken i forveien. I Mobitras har man nemlig mulighet til å gjøre opptak av alle turene man løper, for så å spille dem av mens man løper samme turen senere. Pustingen i nakken er bare lydeffekter som er lagt inn for å gjøre det hele litt mer realistisk."*

Som eksempelet viser, har jeg lagt inn muligheten for å la utøvere gjøre opptak av sine egne treningsturer. Ved å lagre konteksten som registreres under løpeturene, kan man i ettertid fremvise statistikk over utøverens progresjon. Like interessant er muligheten dette også gir for å la utøverne løpe mot sin egen *skygge*. Med "skygge" mener jeg en tenkt virtuell motstander som er basert på et treningsopptak gjort tidligere. Med kontinuerlige oppdateringer over hvordan man ligger an i forhold til en skygge, har man større mulighet til å kontrollere sin egen innsats underveis.



Figur 14: Mobitras i bruk – Figuren viser hvordan bruk av Mobitras ser ut i praksis. Ved hjelp av et bandolær blir GPS-mottaker og PDA festet til kroppen.

Selv om Mobitras ble en del større enn Mocado med tanke på kompleksitet og antall benyttede klasser (Mobitras består av 2834 linjer med kildekode, fordelt på 129 Java-klasser) ble den likevel utviklet i løpet av en kortere periode (fra august til desember 2001). Dette kan forklares med erfaringene jeg gjorde fra utviklingen av Mocado, samt gjenbruk av sentrale løsninger som blant annet behandlingen av GPS-mottakeren.

I dette kapitlet vil jeg beskrive det meste av Mobitras' funksjonalitet som jeg vurderer som relevant for oppgavens tema. Grovt sett kan man dele applikasjonen inn i to deler: Klargjøringen av Mobitras før bruk, og Mobitras som en treningsassistent. Den første tar for seg hvordan man konstruerer løyper og brukerprofiler, mens den andre beskriver hvordan Mobitras benyttes som en assistent under og etter trening. Jeg vil begynne med klargjøringsdelen, og da med konstruksjon av løyper. På samme måte som for Mocado, vil jeg også drøfte mine egne tanker og erfaringer underveis.

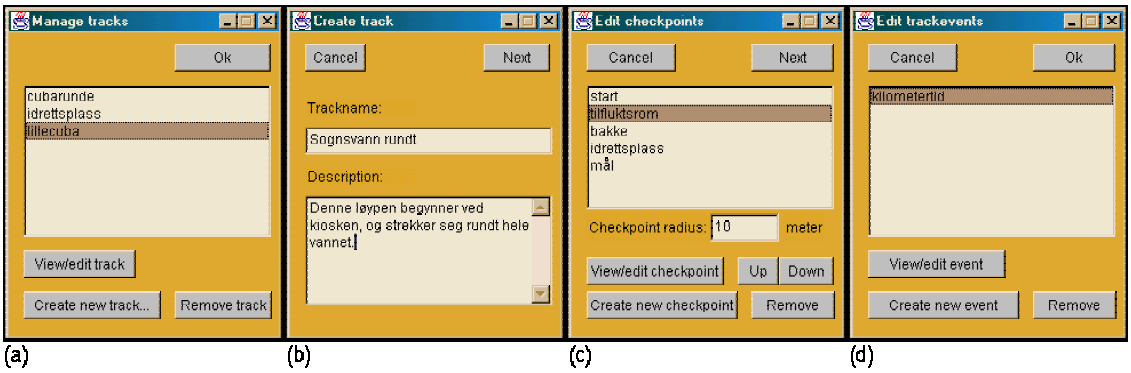
6.1 Konstruksjon av løyper

Før Mobitras kan benyttes, må den klargjøres. Med dette menes at alle løyper en utøver[1] planlegger å løpe, må konstrueres på forhånd (eller kopieres fra noen andre). I tillegg må brukerprofilen som beskriver utøveren spesifiseres. Sett i forhold til oppgavens tema er behandlingen av løyper spesielt interessant; det er først og fremst her kontekstrelevant informasjon konstrueres. Jeg vil av den grunn også begynne med denne delen.

En av årsakene til at man må konstruere løyper på forhånd, er at Mobitras skal ha full kontroll på hvor utøveren begynner et løp, og hvor han går i mål. Med gjenbruk av fastlagte løyper har man også flere muligheter når det kommer til anvendelse av statistikk over utøverens progresjon. Sist, men ikke minst, tillater dette også at Mobitras kan gjennomføre løp sammen med skygger. Straks en løype er ferdigkonstruert vil denne legge seg på PDA-en som en egen fil. Tanken er at flere utøvere kan dele slike løyper med hverandre fritt, og dermed få muligheten til å sammenligne løyperesultater over tid. I tillegg gir dette opphav til en annen spennende tjeneste; ved å dele opptak fra lånte løyper, trenger man ikke bare løpe mot sine egne opptak, man kan også løpe mot opptak gjort av andre.

En løype kan inneholde valgfritt så mange *sekunderingspunkter* som ønskelig (kalt "checkpoints", se Figur 15c). Dette er bestemte punkter langs løypen som utøveren må passere under et løp. I tillegg har man også mulighet til å spesifisere egne *løypehendelser*. Kort fortalt er dette hendelser som ikke er avhengig av hvor man er, men isteden av forhold som hvor langt eller lenge man har løpt. Både sekunderingspunkter og løypehendelser vil jeg forklare nærmere senere. Oppsummert består en løype av et navn, en beskrivelse, et sett med sekunderingspunkter, og et sett med løypehendelser. Jeg vil begynne med å beskrive sekunderingspunktene først.

[1] Jeg velger å bruke benevnelsen "utøver" både for den som konstruerer og løper løypene, da dette gjerne er en og samme person.



Figur 15: Mobitras løypekonstruksjon – Figuren viser hvordan nye løyper blir konstruert. Dialog (a) viser løypene som allerede eksisterer. Ved å velge "create new track…" kommer man videre til (b). Herfra loses man via "next" til dialog (c). Her kan løypens sekunderingspunkter redigeres. Til sist lar dialog (d) brukeren behandle løypens treningshendelser.

6.1.1 Sekunderingspunkter

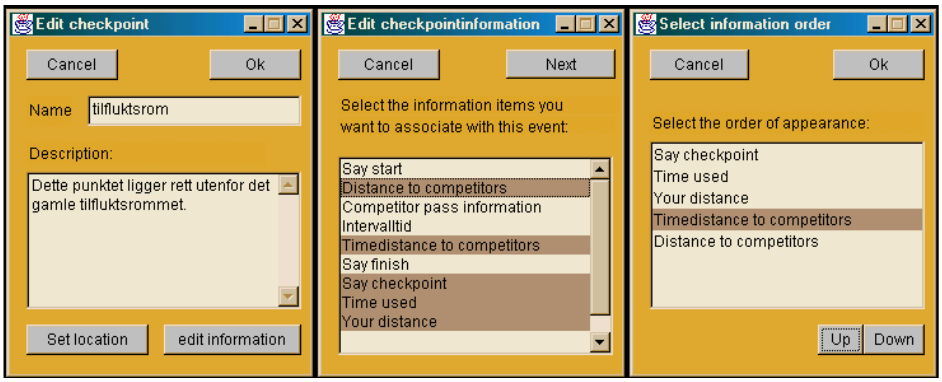
En utøver kan selv kontrollere hvor og når sekunderingsinformasjon skal leses opp under et løp. Dette gjøres ved hjelp av sekunderingspunkter. Disse spres utover hele løypen, og konfigureres av utøveren selv. Konfigureringen bestemmer hva slags informasjon som skal leses opp ved passering (se Figur 16). Slik informasjon kan bestå av tid og tilbakelagte distanse, eller avstand til en eller flere av skyggene. Utøveren står her fritt til å kombinere de ulike parametrene. Et eksempel på hvordan det kan høres ut ved passering av et sekunderingspunkt, er følgende (sammenlign også med Figur 16):

"Checkpoint!"
"Tid: 0.23.18"
"Distanse: 2.8 kilometer."
"Du ligger 0.0.11 foran Odin, 0.7.18 bak Frøya."
"Du ligger 32 meter foran Odin, 1.4 kilometer bak Frøya."

Både Odin og Frøya er her skygger basert på tidligere løp. Hvilke skygger som skal være med under en trening, og hva de skal hete, bestemmes av utøveren på forhånd. Jeg kommer tilbake til skyggene litt senere.

En løype må bestå av minst to sekunderingspunkter, start og mål. Utover dette kan man konstruere så mange punkter man måtte ønske mellom disse. Regelen er at første og siste sekunderingspunkt på listen uansett blir benyttet som henholdsvis start og mål (se Figur 15c). Måten man går frem på for å lage et nytt punkt, er svært likt måten man går frem på for å lage en ny annotering i Mocado; man plasserer seg der man ønsker at punktet skal ligge, og får applikasjonen til å regi-

strere stedet som et nytt sekunderingspunkt[1]. Deretter spesifiseres informasjonen som skal oppleses ved passering, og til sist rekkefølgen på denne (se Figur 16). I motsetning til Mocado har Mobitras ingen kartfunksjonalitet. Av den grunn kan man heller ikke lage sekunderingspunktene uten å fysisk bevege seg ut i løypen[2].



Figur 16: Mobitras sekunderingspunkt – Her vises hvordan man kan konstruere et sekunderingspunkt. I dialogen i midten bestemmes informasjonen som skal leses opp ved passering. Til høyre bestemmes rekkefølgen.

Straks løypen er ferdig blir den lagret på PDA-en. Dette gjør at man ikke trenger å konstruere samme løype mer enn en gang. Man kan imidlertid lage så mange løyper man måtte ønske, selv om disse skulle overlappe plasseringsmessig. Alle blir uansett gjenstand for en egen separat fil.

Tanker og erfaringer om sekunderingspunkter

I utgangspunktet er sekunderingspunktene i Mobitras relativt like Mocados annoteringer. Begge består av kontekstrelevant informasjon, og begge presenteres når brukeren befinner seg innenfor fastsatt areal. I motsetning til Mocado gjør Mobitras imidlertid bruk av mer enn bare lokasjonen for å bestemme utøverens kontekst; det er også et krav at sekunderingspunktene passeres i rekkefølge. Målpunktet kan for eksempel ikke passeres før alle sekunderingspunktene er passert. Noe av idéen bak dette kravet var å hindre utøverne i på en eller annen måte å ta snarveier under løpet. Dette kunne være praktisk om flere utøvere ønsket å sammenligne sine resultater med hverandre i ettertid. Dette var likevel et nedprioritert krav. Det som imidlertid var interessant her, var hvordan Mobitras var avhengig av å ta vare på en historie over aktiverte sekunderingspunkter. Denne informasjonen var viktig for å i det hele tatt kunne bestemme utøverens korrekte kontekst. Uten denne kunne ikke applikasjonen avgjøre om et sekunderingspunkt skulle aktiveres eller ikke, selv om det ble passert. Aktiveringshistorikk var med

[1] Slik som i Mocado, tar også Mobitras imot minst femten posisjoner fra GPS-mottakeren for å avgjøre sekunderingspunktets nøyaktige posisjon.

[2] I Mocado kunne man dette kunne bare ved å bare peke på kartet der annoteringen skulle ligge.

andre ord en del av utøverens historiske kontekst, og viktig kontekstuell informasjon for applikasjonen.

En tredje grunn til at jeg fremsatte kravet om rekkefølge, var av hensyn til implementasjonen. Dette gjorde det mer praktisk å kontrollere om utøveren passerte et sekunderingspunkt. Fordi jeg i applikasjonen kjente rekkefølgen på punktene, trengte jeg kun å holde øye med ett punkt av gangen; det neste på listen. I motsatt fall måtte jeg ha traversert alle kontinuerlig for å avgjøre om noen av de var blitt gyldige. Dette ville vært ugunstig, spesielt om løypen inneholdt mange punkter[1]. En slik test ble foretatt hver gang en ny posisjonsoppdatering ble levert fra GPS-mottakeren, noe som tilsvarte cirka hvert andre sekund. Under uttestting erfarte jeg imidlertid at denne løsningen hadde en litt uheldig konsekvens; om utøveren passerte en sekunderingspunkt uten at dette av en eller annen grunn ble registrert, ville resten av løpet bli ødelagt. Mobitras ville fortsatt vente på passeringen av dette punktet, og derfor ”henge” seg opp på det. Uregistrerte passeringer kunne dessuten relativt lett inntreffe om GPS-forholdene var dårlige.

Ettersom GPS-mottakeren leverte posisjoner med intervaller som beskrevet over, oppsto et annet relatert problem: Hvor stor måtte radiusen til et sekunderingspunkt være dersom en skulle være helt sikker på at en passering ville bli registrert? Fordi posisjonsoppdateringene kom med to sekunders mellomrom, kunne utøveren rekke å forflytte seg relativt langt. Kanskje til og med forbi et helt sekunderingspunkt. Det er vanligvis frustrerende å få en feilaktig beskjed om at ikke alle punktene er blitt passert når man ankommer mål. Man må derfor påse at radiusen blir satt stor nok. Den bør, mer spesifikt, være så stor at det ikke er mulig å passere et helt sekunderingspunkt på under to sekunder.

Det å bestemme en fornuftig radius for sekunderingspunktene kan være relativt greit når alle utøverne er langdistanseløpere. Problemet blir vanskeligere om man også skal inkludere for eksempel syklister. I løpet av tre sekunder vil en syklist med en hastighet på femti kilometer i timen ha tilbakelagt i overkant av førti meter. Med andre ord bør radiusen i så fall dekke rundt tjue meter (som er halvparten av diagonalen som hele sekunderingspunktet spenner ut). Fordi dette kan skape problemer med punkter som ligger for nære, og dermed overlapper, må avstanden mellom punktene tilpasses i forhold til radiusen. Jeg synes det var vanskelig å lage en god løsning som tilfredsstilte flere potensielle idrettsgrener her, og overlot denne avgjørelsen til utøveren (se Figur 15c, ”checkpoint radius”). Dette begrunner jeg med at han er den som sannsynligvis vet hva som passer best. Jo mindre radiusen er, jo nærmere kan punktene ligge. Regelen er at de kan overlappe om nødvendig, så lenge ikke sentrum av det ene punktet kommer innenfor arealet til det andre.

Proporsjonalt med radiusen til et sekunderingspunkt, ble det også vanskeligere å bestemme det nøyaktige tidspunktet for en passering. Skal man være helt nøyaktig (innenfor rammene som blir gitt av GPS) så bør alle sekunderingsdata regi-

[1] JVM-en (Java Virtual Machine) som ble benyttet på Cassiopeiaen eksekverte litt tregt, så jeg forsøkte å unngå slike potensielle flaskehalser i kildekoden.

streres akkurat i det sentrum passeres. Problemet oppstår imidlertid når oppdateringene fra GPS-mottakeren inntreffer med et lite intervall. Om man for eksempel benytter første registrerte posisjon innenfor sekunderingspunktet, kan tidsmålingen bli relativt unøyaktig (spesielt når radiusen er stor, se Figur 17).

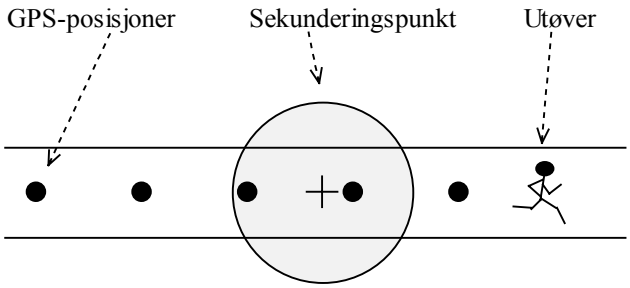
Ettersom jeg vurderte disse feilmålingene til å være alvorlige for applikasjonens brukbarhet, så jeg meg nødt til å implementere ekstra funksjonalitet som rettet på dette. Grovt skissert virker algoritmen jeg kom frem til på følgende måte[1]: Applikasjonen tar vare på posisjonene som blir mottatt like før og like etter sentrum av sekunderingspunket (henholdsvis a, b, og s. Se for deg disse som de tre punktene innenfor radiusen i Figur 17). Ved hjelp av disse kan den videre bestemme avstanden fra *a* til *b*, og fra *a* til *s*. Fordi applikasjonen vet tidspunktet når både *a* og *b* inntraff, kan den også regne seg frem til tidspunktet når *s* ble passert. Dette gjøres ved å regne ut prosentvis avstand mellom *a* og *s* i forhold til avstanden mellom *a* og *b*, og deretter multiplisere denne verdien med avstanden i *tid* mellom *a* og *b*. Denne algoritmen blir mest nøyaktig dersom sekunderingspunktet ligger på en rett strekning, og dersom utøveren passerer det med en jevn hastighet.

Passeringsalgoritmen er avhengig av å bruke en måling som kommer etter at utøveren har passert midten, og da oppstår det en liten forsinkelse. Dette er tiden det tar fra sekunderingspunktet passeres, til sekunderingsinformasjonen er regnet ut og leses opp. Om man ikke er klar over dette, kan man villedes til å tro at verdiene er blitt registrert for sent (fordi de leses opp etter passeringen). Analogt må utøveren også fullføre noen meter ekstra etter at han har kommet i mål; applikasjonen er på samme vis avhengig av å registrere en observasjon etter målpasseringen.

Ved passering av et sekunderingspunkt blir utøverens treningsdata (eller sekunderingsinformasjon) lest opp. I utgangspunktet hadde jeg hardkodet hva dette skulle være, men fant etter hvert ut at det ville være mer praktisk om utøveren kunne bestemme denne informasjonen selv. Dette gjaldt også rekkefølgen på hvordan den skulle bli lest opp. Grunnen til dette viste seg gjerne når man brukte applikasjonen i praksis; om man stadig fikk opplest en masse informasjon som egentlig ikke var ønsket, ble dette et irritasjonsmoment under løpet. Spesielt var dette merkbart da talesystemet besto av innspilte ord og tall som Mobitras satte i sammen til hele setninger[2]. Om for eksempel fire skygger var med i løpet, kunne opplesningen etter en passering ta over et halvt minutt (om differansen i tid og lengde til disse skulle være med). Da kunne det være greit å sløyfe uinteressant informasjon, eller i alle fall få lest opp sin egen tid og distanse først.

[1] Denne algoritmen er ikke viktig å forstå for resten av oppgaven.

[2] Dette ble gjort fordi jeg ikke hadde noen tilgjengelig talesynthesizer som kunne benyttes på PDA-en



Figur 17: Mobitras ved passering av sekunderingspunkt – Figuren viser hvordan posisjonsoppdateringene kan komme litt ugunstig i forhold sentrum av et sekunderingspunkt. Utøveren, som løper på veien mot høyre, har akkurat passert punktet. GPS-posisjonene viser hvor på strekningen han var da han mottok oppdateringene.

6.1.2 Treningshendelser

> *"For hver kilometer Anne tilbakelegger gjennom skogen, aktiveres en egen treningshendelse. Hendelsen har navnet "kilometertid", og denne er uavhengig av sekunderingspunktene. For eksempel leses følgende melding opp i det hun runder to og en halv mil: 'Kilometertid: 0.9.13'."*

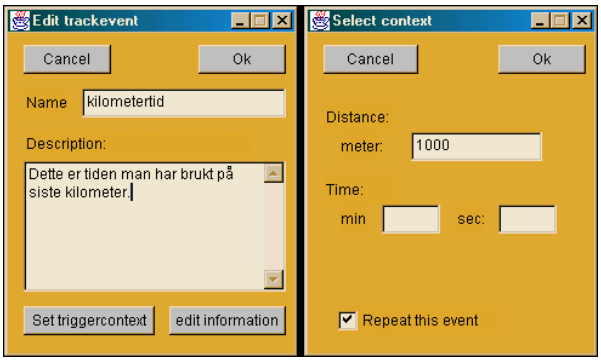
En utøver har mulighet for å konstruere det jeg har valgt å benevne som *treningshendelser*. En treningshendelse kan med god grunn sammenlignes med et sekunderingspunkt; begge består av informasjon knyttet til en kontekst. Forskjellen er at kontekstspesifikasjonen hos et sekunderingspunkt består av et areal og et krav om rekkefølge. Hos en treningshendelse består kravet isteden av en tilbakelagt tid eller avstand. I Annes tilfelle ble for eksempel kilometertiden (tiden brukt på å løpe siste kilometer) realisert på denne måten.

Informasjonen man kan feste til en treningshendelse er relativt lik den man kan feste til et sekunderingspunkt. Faktisk er det de samme dialogvinduene som også blir vist når man konstruerer dem. Forskjellen, naturlig nok, er hvordan man går frem for å spesifisere aktiveringskonteksten. Dette gjøres i et eget dialogvindu hvor man kan føre opp antall meter og/eller sekunder som skal være tilbakelagt, og eventuelt om hendelsen skal repeteres kontinuerlig som et slags intervall (se Figur 18).

Tanker og erfaringer om treningshendelsene

Muligheten for at en applikasjon ikke tilfredstiller sluttbrukernes egentlige behov er tilstede om de ikke inkluderes under utviklingen [Sommerville 1998]. Riktignok hadde både jeg og andre involverte i Mobitras trent litt fra tid til annen, men aldri under mer ordnede forhold (slik som for eksempel i en løpeforening). Jeg valgte derfor å ta kontakt med Idrettshøyskolen i Oslo for å undersøke om de kunne tenke seg å delta i prosjektet. Håpet var at de kunne bidra under uttesting-

en. Det siste ble det dessverre aldri noe av, med jeg fikk i stand et møte hvor Mobitras ble presentert og diskutert. Det var også her kravet om ”kilometertid” ble trukket frem. Dette, mente de, var kanskje den aller viktigste parameteren langdistanseløpere faktisk var opptatt av.



Figur 18: Mobitras treningshendelser – Her vises hvordan treningshendelsen ”kilometertid” konstrueres. Ved å velge ”Set triggercontext” i dialogen til venstre, dukker dialogen til høyre opp. Her kan aktiveringskonteksten spesifiseres.

Etter møtet fant jeg raskt ut at tilbakemelding om kilometertid ikke lot seg løse i Mobitras. Sekunderingspunktene ble assosiert til en lokasjon, og ikke utøverens tilbakelagte avstand. Ettersom denne parameteren ble fremstilt som et klart ønske fra sluttbrukerne, valgte jeg å implementere en løsning også for dette. Imidlertid vurderte jeg kilometertiden til å være litt for spesifikk til at den strengt tatt var interessant for denne hovedoppgaven. Jeg valgte derfor å foreta noen ekstra generaliseringer over dette, slik at funksjonaliteten sto mer i stil med oppgavens tema. Resultatet ble treningshendelsene.

En spesielle egenskap hos treningshendelsene er hvordan applikasjonen må gå frem for å avgjøre om noen av dem er blitt gyldige. Utøverens tilbakelagte distanse blir ikke registrert ved hjelp av GPS-posisjonene direkte. Man har behov for å kjenne til hans avstand, og ikke hans posisjon. Posisjonene må derfor omformes ved hjelp av en egen komponent. Denne virker som et mellomledd mellom applikasjonen og komponenten som kommuniserer med GPS-mottakeren, og omformer posisjoner over tid til tilbakelagt distanse. Distanseverdiene, igjen, anvendes som kontekstuell informasjon på lik linje med posisjonene.

Etter at treningshendelser var blitt implementert, så jeg meg ferdige med Mobitras. Likevel var det flere forhold som med fordel kunne ha blitt endret (eller implementert). Det første har med hvem som ”eier” treningshendelsene. Slik Mobitras fungerer i dag, assosieres disse til løypene. Siden man likevel kan vurdere disse som uavhengig av løypene, burde de muligens heller vært assosiert til utøverne. Dette ville eliminert behovet for å konstruere like hendelser for hver av løypene de benytter.

Jeg så også en generalisering når det gjaldt behandling av skygger. Man får alltid beskjed om når man passerer, eller blir passert av en av skygge. Fordi dette er

hendelser som inntreffer underveis i løpet, kan de også vurderes som treningshendelser. Informasjonen som leses opp kan i så fall bli knyttet til situasjonen hvor brukeren passerer eller blir passert av en skygge. Det siste kan også argumenteres for å tilhøre utøverens kontekst. I så fall kunne skyggene ha blitt realisert som en utvidelse av treningshendelsene.

Det er også hensiktsmessig å diskutere hvorvidt passering av en skygge er presentasjon av kontekst eller kontekstrelevant informasjon. Grensen her er vag, og avhenger gjerne av hvordan man anvender informasjonen. Det å passere en skygge kan, som nevnt, argumenteres for å være en del av utøverens kontekst. Det samme gjelder kilometertiden. Imidlertid blir de begge i Mobitras behandlet som kontekstrelevant informasjon.

6.2 Behandling av utøvere

For å la flere utøvere benytte samme utstyr, tillater Mobitras flere brukerprofiler (se Figur 19). En brukerprofil består ikke av annet enn utøverens navn, og anvendes kun for å navngi treningsopptakene hver enkelt utøver produserer. Dette gjør det enklere å skille opptakene fra hverandre i ettertid (spesielt om de utveksles mellom utøverne).

Figur 19: Mobitras brukerspesifikasjon – Her vises dialogen som lar brukeren legge til, og fjerne, brukerprofiler.

Denne funksjonaliteten berører ikke oppgavens tema i vesentlig grad, og jeg velger å ikke gå mer i detalj på dette. Jeg vil imidlertid inkludere slike profiler i neste seksjon, når jeg beskriver hvordan Mobitras kan anvendes som en treningsassistent.

6.3 Mobitras som treningsassistent

Etter at løyper og brukerprofiler er opprettet, er Mobitras klar for treningsanvendelse. Denne fasen består av tre deler: Treningsoppsett, gjennomføring, og treningsdagbok. Med andre ord: Før, under, og etter en trening. Jeg vil forklare disse delene i rekkefølgen de inntreffer, og begynner derfor med treningsoppsettet.

6.3.1 Treningsoppsett

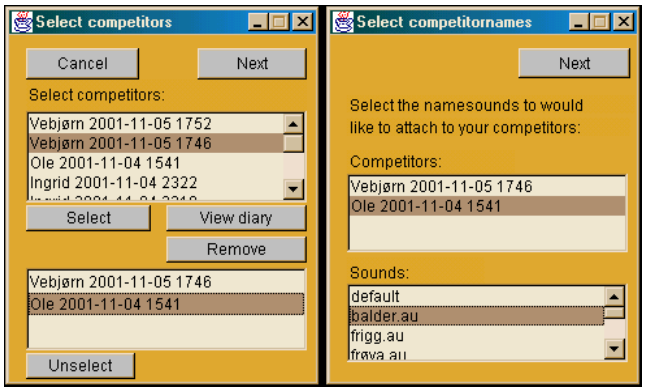
Første skritt ved påbegynnelse av en ny trening er å forklare Mobitras hvem du er, hvor du ønsker å trene, og hvem du ønsker å trene med. For å gjøre denne prosessen rask og enkel blir man loset gjennom en rekke dialogvinduer, og praktisk talt frem til start.

Første stopp i oppsettet er valg av brukerprofil. Flere utøvere kan bytte på å anvende PDA-en, og Mobitras kan av den grunn også ivareta flere slike profiler. Dialogen som setter oppsettet i gang fremkalles ved å velge ”Begin new training” fra hovedmenyen (se Figur 20). Fra profilvalg går veien videre til valg av løype[1], og deretter til valg av skygger (se Figur 21). Her vises alle registrerte opptak, både egenproduserte og importerte, som er utført på den valgte løypen. Utøveren kan omforme til skygger så mange av disse som han ønsker, men de som velges må imidlertid få et kallenavn slik at Mobitras kan referere til de under løpet.

Figur 20: Mobitras hovedmeny – Dialogen som presenteres når applikasjonen startes.

Når informasjonen ovenfor er lagt inn, og utøveren har innfunnet seg på startplass (som egentlig er første sekunderingspunkt), er Mobitras klar for start. Straks utøveren gir tegn (som jeg vil komme til nedenfor) settes løpet i gang.

[1] Disse dialogene er de samme som anvendes ved bruker-, og løypekonstruksjonen. Det eneste som varierer her er at hver enkelt har fått en knapp benevnt ”*next*” som tar brukeren videre i oppsettet.



Figur 21: Mobitras skyggedialog – Dialogen til venstre viser hvordan man velger hvilke skygger man vil ha med under en trening, mens dialogen til høyre viser hvordan man kan gi disse et midlertidig navn. Alle treninger ligger sortert etter dato, med de nyeste øverst.

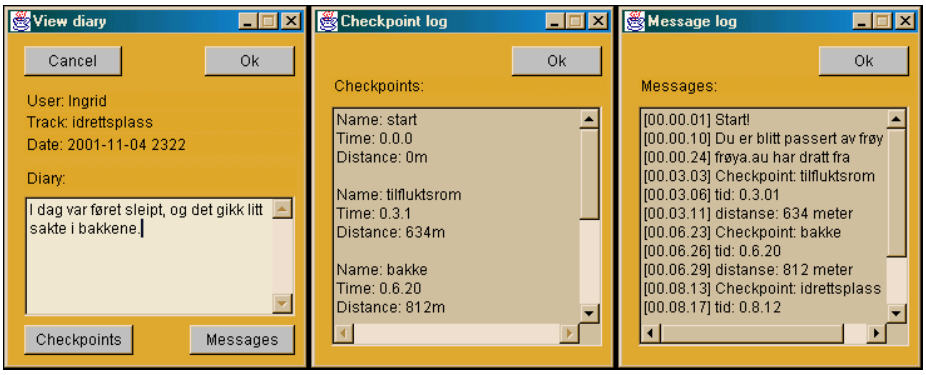
6.3.2 Gjennomføring

Underveis i et løp er det tre tilfeller hvor Mobitras leser opp informasjon: Ved passering av sekunderingspunkter, ved aktivering av treningshendelser, eller ved passering av en skygge (eller hvis skyggen eventuelt passerer utøveren). I tillegg blir det gitt beskjed om noen av skyggene løper i mål. Før en eventuell passering inntreffer, vil også utøveren bli underrettet om dette på forhånd. Er han er i ferd med å bli forbiløpt av en skygge, vil for eksempel følgende meldinger kunne bli lest opp (etter hvert skyggen avanserer):

Melding fra Mobitras	Avstand til skygge
”Balder nærmer seg bakfra!”	5-10 meter bak
”Du ligger likt med Balder!”	+/- 5 meter
”Du har blitt passert av Balder!”	10-15 meter foran
”Balder har dratt fra!”	Mer enn 15 meter foran

6.3.3 Treningsdagbok

Når siste sekunderingspunkt passeres, er utøveren kommet i mål. Straks de nøyaktige målpasseringsverdiene er regnet ut (jamfør tanker og erfaringer under 6.1) går applikasjonen inn i treningsdagboken. Her kan utøveren blant annet fylle inn treningsnotater og tanker om hvordan gjennomføringen forløp seg (se Figur 22). Her kan han også få en oversikt over hvilke verdier som ble registrert for sekunderingspunktene, samt alt av meldinger som Mobitras leste opp underveis.



Figur 22: Mobitras treningsdagbok – Dialogen til venstre viser utøverens egne treningsnotater for den gjennomførte treningen. I midten vises loggen over sekunderingspunktene, mens til høyre ses de oppleste meldingene.

Når brukeren har gjort seg ferdig med dagboken og valgt ”ok”, blir treningen oppbevart som en selvstendig fil. Denne vil inkludere alt av kontekstdata, meldinger, og lignende, som ble registrert under treningen. Som tidligere nevnt, tillater dette brukerne å utveksle opptak. På den måten kan de også konkurrere med hverandres skygger. Man kan når som helst få frem dagboken igjen ved å velge ”Diary” fra hovedmenyen (se Figur 20). I så tilfelle dukker det opp en ny dialog hvor alle de gjennomførte treningene (både egne og importerte) ligger sortert etter løype og dato.

6.3.4 Tanker og erfaringer om Mobitras som en treningsassistent

Mobitras kunne aldri blitt hensiktsmessig å anvende i praksis om utøveren var nødt til å kikke på en skjerm mens han løp. Jeg var også av den oppfatning at det ville bli upraktisk om han måtte holde PDA-en i hånda. Jeg ønsket derfor å tilrettelegge applikasjonen slik at utstyret kunne ”gjemmes bort” under løpet (festes i beltet, legges ned i en lomme, eller lignende). Første krav gikk derfor på kommunikasjon. All kommunikasjon skulle foregå ved hjelp av lyd. Dette gjaldt både meldingene som kom fra Mobitras, men også eventuelle kommandoer fra utøveren. Det siste kravet viste seg imidlertid som vanskelig. Både utstyr og avsatt tid for Mobitras hadde sine klare begrensninger. Heller ikke normal lydavspilling var enkelt. Faktisk var ikke lydavspilling i JDK 1.3 støttet for annet en ”applets”. Etter litt leting fant jeg likevel frem til en skjult, ikke-dokumentert, pakke (sun.audio) som tillot avspilling av lyd i au-format[1]. Jeg valgte derfor en løsning hvor Mobitras kommuniserer ved hjelp av lyd, mens utøveren anvender penn og skjerm.

En annen funksjon jeg la inn, var nedtelling til start. Slik applikasjonen var utformet i begynnelsen ble løpet igangsatt umiddelbart etter at utøveren hadde

[1] Jeg benyttet også lyd i Mocado. Da anvendte jeg imidlertid en løsning hvor jeg startet en ny prosess med WinCE-plattformens eget lydavspillingsprogram. Denne løsningen ble imidlertid for snever for Mobitras. Her var mer kontroll over selve lydavspillingen påkrevd.

trykket på knappen ”start”. Dermed måtte han hekte fra seg PDA-en i beltet samtidig som løpet begynte. Dette var særdeles ugunstig. Derfor inkluderte jeg også til slutt en egen nedtellingsfunksjon. Denne talte ned et visst antall sekunder (muntlig) før løpet faktisk ble igangsatt (se Figur 23). Om utøveren skulle forlate området før denne tiden var omme, vil dette bli tolket som tjuvstart. Løpet ble i så fall stoppet.

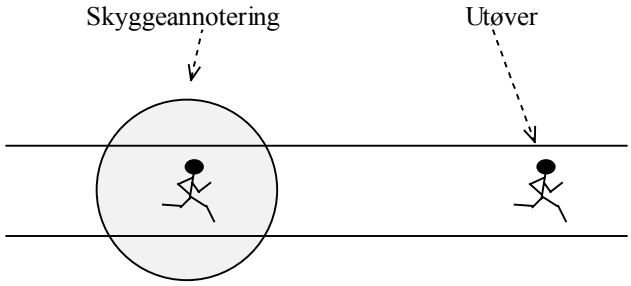
Figur 23: Mobitras ved start – Dialogen som vises når brukeren befinner seg på startsreken og er klar for start.

Flere forhold med Mobitras gjør applikasjonen interessant med tanke på hovedoppgavens tema. Alle forholdene dreier seg om hvordan kontekst og kontekstrelevant informasjon blir anvendt. Hvis man ser på Mocado, så forblir annoteringene her uforanderlige etter at de er blitt forfattet og lagret (dersom man ikke eksplisitt endrer på dem i ettertid). Dette gjelder både aktiveringskonteksten (som besto av en lokasjon og en radius), og dens assosierte informasjon. I Mobitras er dette annerledes. Informasjonen som skal presenteres ved passering av et sekunderingspunkt kan ikke fullt og helt bestemmes på forhånd. Man kan bare angi hva slags type informasjon som skal presenteres, ikke innholdet. Dette avgjøres av applikasjonen idet sekunderingspunktet passeres. For eksempel kan man avgjøre at et sekunderingspunkt skal presentere ”tilbakelagt distanse”. Men hvilken verdi som leses opp, kan ikke avgjøres før utøveren passerer punktet. Med andre ord er denne informasjonen dynamisk, og krever beregninger før den kan presenteres. Det samme forholdet gjelder også for treningshendelsene.

Et annet forhold som kan observeres, er at informasjonen hos sekunderingspunktene stort sett bare består av kontekstuell informasjon. Både kilometertid og tilbakelagt distanse kan for eksempel vurderes som utøverens kontekst. Kontekstuell informasjon presenteres med andre ord direkte til brukeren, men da som kontekstrelevant informasjon (den er bundet til et sekunderingspunkt eller en treningshendelse). I disse tilfellene fungerer kontekstuell informasjon med andre ord også som kontekstrelevant informasjon. Dette er et forhold jeg vil drøfte også senere i oppgaven.

Det er interessant å se hvordan Mobitras behandler skygger. Når utøveren kommer i nærheten av en skygge, enten forfra eller bakfra, forteller Mobitras utøveren om skyggen. Dette er omtrent det samme som når utøveren passerer et sekun-

deringspunkt. Forskjellen er at mens et sekunderingspunkt står stille, beveger skyggen seg. Det vil si, skyggen kan vurderes som en bevegelig annotering. Hvis man følger denne betraktningen videre, består en skyggeannotering av skyggens dynamiske posisjon sammen med en radius på ti meter (se Figur 24). Når utøveren kommer innenfor dette arealet (enten ved at han løper inn i skyggen, eller at skyggen "løper" inn i han) aktiveres informasjonen som forteller om skyggens nærvær.



Figur 24: Mobitras og skygger – Her ser man utøveren med en skygge like bak. Skyggen representeres ved hjelp av en annotering som oppdateres i henhold til skyggens egen posisjon.

Hos skyggeannoteringene er det først og fremt aktiveringskonteksten som endrer seg: Hvor skyggen ligger plassert rent geografisk, avhenger av hvor lenge treningen har pågått. I begynnelsen av løpet vil den ligge i begynnelsen av løypa, og etter hvert som tiden går vil den forflytte seg nærmere mål. Den er jo et opptak av en tidligere trening. Aktiveringskonteksten til en skyggeannotering inkluderer derfor både en tid og lokasjon. Begge endrer seg underveis.

Illustrasjonen i Figur 24 viser egentlig bare halve sannheten. Som nevnt aktiveres forskjellig informasjon etter hvert som skyggen nærmer seg. Når den er fem til ti meter unna får man beskjed om at den er like bak (eller foran), mens når det er mindre en fem meter får man beskjed om man ligger likt. Dette indikerer at det ikke bare er én annotering det er snakk om, men to. Den første hadde en radius på ti meter, mens den andre hadde en radius på fem. Det er heller ikke entydig hvilken informasjon som skal presenteres når disse aktiveres. Dette avhenger isteden av utøverens kontekst. Hvis han ankom annoteringen bakfra, så fikk han beskjed om at han var i ferd med å løpe opp skyggen. Om han isteden ankom forfra (slik at skyggen var i ferd med å ta han igjen) fikk han beskjed om at skyggen lå like bak. Med andre ord var informasjonen dynamisk, og dens innhold ble bestemt av retningen til utøveren før aktiveringen.

Et problem som oppsto med skyggene og deres annoteringer i praksis, var når utøveren lå like bak (eller foran) en av skyggene. Da inntraff grenseproblematikken: Utøveren løp inn og ut av skyggeannoteringen flere ganger etter hverandre. Dette skjedde gjerne om man lå like bak annoteringen, men ikke klarte å øke tempo nok til at man kom ordentlig "inn" i den. Dette var analogt med problemet i Mocado; annoteringer som ble aktivert og deaktivert unaturlig hyppig. For å bøte på dette i Mobitras, introduserte jeg et nytt felt i skyggeannoteringene som

jeg kalte for *deaktiveringskontekst*. Mens aktiveringskonteksten spesifiserte når informasjonen i skyggeannoteringen var gyldig, beskrev deaktiveringskonteksten når den kunne vurderes som ugyldig igjen. For eksempel valgte jeg å sette deaktiveringsradiusen hos skyggeannoteringene til å være fem meter større en aktiveringsradiusen. På den måten var utøveren nødt til å komme hele fem meter unna (og tilbake igjen) før annoteringen kunne bli aktivert på nytt etter en deaktivering. Dette løste til slutt grenseproblematikken på en tilfredsstillende måte.

6.4 Oppsummering

Grunnen til at jeg valgte å gå i gang med en ny eksempelapplikasjon etter Mocado, var for å undersøke hvordan en litt annerledes kontekstoppmerksom applikasjon kunne fortone seg. Jeg ønsket å se om andre og nye krav ville gjøre seg gjeldene om applikasjonen ikke var en eller annen form for turistguide. I ettertid mener jeg også å ha lykkes bra med det. Det er åtte forhold som bør trekkes frem fra Mobitras som har betydning for hovedfagsoppgaven:

Bruk av historisk kontekst – For å avgjøre om et sekunderingspunkt skulle aktiveres, var Mobitras avhengig av å kjenne til hvilke andre sekunderingspunkter utøveren allerede hadde passert. Dette var en del av hans kontekst. Jeg benevnte dette som hans ”aktiveringshistorikk”, og vurderte denne informasjonen som viktig for applikasjonen.

Nøyaktighet – Et ideelt utgangspunkt for å avgjøre når en utøver passerte et sekunderingspunkt, ville ha vært om punktet nærmest var uten utstrekning. Med det samme utøveren tråkket nøyaktig på dette, ville det også bli aktivert. Likevel lot ikke Mobitras seg løse på denne måten. Dette kom av GPS-mottakeren: Målingene herfra kom alt for sjeldent. Dermed fikk jeg problemer både med å avgjøre om punktet faktisk ble passert, og i så fall hva som var nøyaktig passeringstid. Dette problemet kan også generaliseres til å gjelde for andre sensorer. Om målingene fra disse er for grovkornede i forhold til intervallene som anvendes i aktiveringskontekstene, kan man risikere at gyldig kontekstrelevant informasjon ikke blir registrert. I Mobitras måtte jeg av den grunn implementere egen funksjonalitet for å avgjøre nøyaktig passeringstid.

Aktiveringskontekst uavhengig av sted – Mobitras benyttet også treningshendelser. Det spesielle med disse var aktiveringskonteksten. I motsetning til annoteringer og treningshendelser besto ikke denne av et sted, men isteden av utøverens tilbakelagte tid og distanse. Like fullt ble de behandlet som kontekstrelevant informasjon på lik linje med sekunderingspunktene.

Omforming av kontekst – Det var også interessant å kikke på parameteren ”tilbakelagt distanse”. Denne kunne spesifiseres som en del av aktiveringskonteksten til treningshendelsene. Verdien her ble likevel ikke bestemt av noen sensor direkte. Isteden ble den utregnet underveis ved hjelp av kunnskap om utøverens

posisjonering bakover i tid. GPS-koordinatene som Mobitras mottok ble med andre ord omformet til en ny og annerledes kontekstuell informasjon.

Alternative presentasjonsformer – Selv om jeg benyttet lydavspilling også i Mocado, anvendte jeg dette i større grad i Mobitras. All informasjon som ble presentert for utøveren under et løp, ble gitt ved hjelp av lydmessige tilbakemeldinger.

Dynamisk kontekst og informasjon – Kanskje aller mest interessant ved Mobitras var det faktum at verken aktiveringskontekst eller kontekstrelevant informasjon i mange tilfeller kunne avgjøres på forhånd. Man kunne bare angi hva slags type informasjon som skulle presenteres, ikke innholdet. Dette gjaldt spesielt aktiveringskonteksten hos skyggeannoteringene; deres plassering var avhengig av tiden utøveren hadde tilbakelagt (jo lengre tid, jo nærmere mål).

Uavhengig informasjon – Det er ikke alltid slik at kontekstrelevant informasjon trenger å beskrive noe i relasjon til stedet hvor det ligger plassert. Treningshendelsene er et godt eksempel på dette. Til tross for at disse kunne presentere informasjon som en følge av at utøveren hadde løpt en kilometer, kunne opplysningene de kom med likevel dreie seg om noe helt annet. For eksempel kunne de settes til å bare spille av litt jubelbrus. Slik informasjonen trenger med andre ord ikke være basert på brukerens kontekst.

Separat deaktiveringskontekst – Til sist demonstrerte Mobitras bruk av separate deaktiveringskontekster. Dette var et nødvendig krav for skyggeannoteringene. For å unngå at en utøver kunne ligge på grensen til en skygge, og dermed aktivere og deaktivere denne om hverandre, var jeg nødt til å skille klarere mellom aktivering og deaktivering. Løsningen ble å ta i bruk en separat deaktiveringskontekst hvor en fem meter større radius ble satt i forhold til aktiveringskonteksten. Dette tvang utøveren minst fem meter unna (og tilbake igjen) før en reaktivering kunne inntreffe.

Selv om Mobitras tilfredsstilte mange av mine fremsatte krav, var det likevel mye som sto ugjort eller ikke virket som det skulle da jeg avsluttet prosjektet. Ett av disse var løping på bane. Dette var et krav jeg ikke hadde tenkt på, men som ble fremsatt under møtet med Idrettshøyskolen. Slik Mobitras var konstruert, lot det seg ikke gjøre å legge inn antall runder en løype skulle løpes. En dårlig løsning var derfor å konstruere opp like mange runder som man ønsket å løpe på forhånd (slik at man isteden fikk én lang løype). Dette betydde dupliserte sekunderingspunkter for hver eneste runde. En bedre løsning ville ha vært å benytte én runde, og isteden oppgitt antall ganger man ønsket å repetere denne. Prosjektets tidsramme satte imidlertid stopp for dette.

Jeg planla også å inkludere egne menyer med statistikk. Etter hvert vurderte jeg imidlertid denne delen som en avsporing fra oppgavens egentlige tema. Jeg valgte av den grunn å heller konsentrerte meg mer om funksjonalitet rundt behandlingen av kontekstoppmerksomhet og kontekstrelevant informasjon. Jeg mener likevel at bruk av statistikk ville ha beriket Mobitras som treningsassistent i vesentlig grad. Derfor har jeg også tilpasset applikasjonen innad med tanke på dette.

En tredje funksjonalitet som lenge var aktuell, var å tilby utøveren en egen virtuell treningsassistent. Assistentens oppgave skulle være å monitorere løpene underveis, slik at den kunne komme med råd og innspill både under og etter en trening. Også her anslo jeg dette til å være en avsporing i forhold til oppgavens egentlige tema.

6.5 Ergonomiske forhold rundt Mocado og Mobitras

Et av utgangspunktene for mobilitetsbaserte kontekstoppmerksomme applikasjoner er å støtte mobile brukere med informasjon eller funksjonalitet relatert til situasjonen de befinner seg i [Pascoe 1998]. Når mengden utstyr likevel overstiger en viss grense (slik som det kan føles når man bærer rundt på en PDA med to meter ledning og en tilkoblet GPS-mottaker) kan man stille seg spørsmålet i hvilken grad man egentlig er mobil. På et vis har man redusert denne evnen på bekostning av tjenestene man får tilbake. Chen hevder at sensorene som benyttes bør være små og nesten usynlige, og at brukeren må kunne legge vekk, eller ta i bruk, disse etter som det passer [Chen et al. 2000]. Selv om slike forhold er viktige suksesskriterier om både Mocado og Mobitras skulle bli benyttet kommersielt, har jeg likevel ikke vektlagt dette i oppgaven. Jeg har kun studert hva slags funksjonalitet man i slike applikasjoner har behov for med tanke på et rammeverk.

Kapittel 7

Anvendelse av kontekstoppmerksomhet

Etter å ha presentert Mocado og Mobitras, samt belyst fire andre eksempelapplikasjoner, vil jeg nå identifisere betydningsfull funksjonalitet blant slike applikasjoner mer generelt. Jeg vil med andre ord analysere hvordan kontekstoppmerksomhet og kontekstrelevant informasjon blir anvendt. Jeg vil se på bruk av kontekstoppmerksomhet i dette kapitlet, og bruk av kontekstrelevant informasjon i neste. Målet for begge kapitlene er å oppnå en kravspesifikasjon for et rammeverk. Denne vil bli konkretisert i Kapittel 9.

Jeg har valgt å skille analysen av kontekstoppmerksomhet inn i to deler: Anvendelse i forhold til applikasjonene, og anvendelse i forhold til brukerne. I den første delen vil jeg drøfte hvordan applikasjonene håndterer kontekst for å tilby kontekstoppmerksomhet. Dette inkluderer hva *slags kontekst* de benytter, og *hvordan* de registrerer den. Dette er forhold som i stor grad bør skjermes fra applikasjonsutviklerne [Dey 2000]. I den andre delen vil jeg drøfte applikasjonene fra brukernes ståsted. Hva slags behov og ønsker har de med tanke på kontekstoppmerksomhet? Begge disse delene vurderer jeg som relevante å belyse når kravene for et rammeverk skal identifiseres.

7.1 Anvendelse i forhold til applikasjonene

For at en applikasjon skal kunne være kontekstoppmerksom, må den ha mulighet for å måle signifikante attributter omkring brukerens kontekst [Dix et al. 2000]. Slik måling vil jeg benevne som kontekstregistrering. Det er minst to spørsmål som gjør seg gjeldene rundt dette [Dey 2000]: Hva slags kontekstuell informasjon gjør applikasjonene bruk av, og hvordan registrerer de den? Disse to spørsmålene er gjennomgangstemaet for dette avsnittet.

Jeg har identifisert fem metoder for registrering av kontekst: Bruk av sensorer, kommunikasjon mellom entiteter, manuell spesifikasjon, historikk, og kontekst-

utledning. Disse vil jeg drøfte under. For hver metode vil jeg også gjøre rede for hva slags kontekstuell informasjon som metoden registrerer.

7.1.1 Anvendelse av sensorer

Som nevnt tidligere er det fordelaktig å la applikasjoner registrere kontekst automatisk uten hjelp fra brukeren [Dey 2000]. En av metodene jeg har identifisert for å realisere dette, er bruk av sensorer. Andersson beskriver disse slik:

> *"A sensor can be described as a system, which detects signals or objects of a specific type. Depending on, if the sensor is a radar, TV-camera, infrared camera, acoustic sensor, transponder, passive radar or any of the many other types of sensors, it provides some information about events or environmental settings."*
>
> – [Andersson et al. 1999, side 42].

Jeg vil skille mellom to ulike sensorsystemer: *Interne* og *eksterne*. Med et internt sensorsystem mener jeg egne komponenter innad i applikasjonen som registrerer kontekstuell informasjon fra applikasjonen eller operativsystemet. Opplysninger om tid og dato, tilgjengelig nettverksforbindelse, minnekapasitet, og lignende, er eksempler på slik informasjon. Et eksternt system, derimot, består av komponenter som kommuniserer med fysiske eksterne sensorer. I Mocado ble et slikt system anvendt for å registrere brukerens posisjon (ved hjelp av en GPS-mottaker).

Eksterne sensorsystemer av interesse

Hvilke eksterne sensorsystemer som benyttes blant eksempelapplikasjonene, varierer. Alle har imidlertid et system for å bestemme brukerens posisjon. Likevel varierer det relativt mye når det gjelder hva slags posisjoneringsløsninger som benyttes. Mens Field assistant anvender GPS, benytter Abowd i Cyberguide et egenkonstruert system med fjernkontroller. Davies (The Lancaster Guide) anvender isteden en løsning med et trådløst nettverk. I både Mocado og Mobitras anvender jeg GPS.

Det at alle eksempelapplikasjonene benytter opplysninger om brukerens posisjon, er i seg selv ikke overraskende. Tross alt er lokasjon vurdert som en hovedparameter for mobilitetsbaserte applikasjoner [Dix et al, Salber et al] (jamfør 3.4). Jeg ser det heller ikke som spesielt at dette er den eneste kontekstparameteren som applikasjonene registrerer ved hjelp av et eksternt sensorsystem. Både Dey, Pascoe, og Salber hevder at det er problemene knyttet til anvendelse av sensorer som medfører at kontekstoppmerksomhet så sjeldent blir benyttet [Dey 2000, Pascoe 1998, Salber et al.1998]. En observasjon jeg likevel vektlegger er at det tas i bruk

mange ulike posisjoneringsløsninger, spesielt innendørs. Derfor oppfatter jeg det også som problematisk å forutse hva slags posisjoneringssystemer som kan bli benyttet i fremtidige applikasjoner. Det er dessuten naturlig å tro at nye og bedre posisjoneringsteknologier vil gjøre seg gjeldene etter hvert.

For utendørsposisjonering er situasjonen annerledes. Her blir GPS tatt i bruk av nesten samtlige (unntaket er The Lancaster Guide). Jeg vil av den grunn argumentere for at dette systemet bør gis spesiell støtte i et rammeverk, kanskje i form av bibliotekskomponenter (jamfør Kapittel 2). Selv om brukerens posisjon er den eneste kontekstparameteren som blir registrert ved hjelp av et eksternt sensorsystem, har jeg likevel tro på at også andre typer sensorer kan være relevante å benytte om forholdene legges bedre til rette. Brown argumenterer blant annet for at bruk av lydgjenkjenning kunne være en interessant kontekstuell parameter for en rekke applikasjoner [Brown 1995]. Dette kunne for eksempel ha blitt registrert ved hjelp av en mikrofon. Selv mener jeg at både temperatur (ved hjelp av et termometer), puls (en pulsmåler), retning (et digitalt kompass) og lignende også er relevante parametere som kan registreres på lignende vis. Spesielt de to siste (puls og retning) kunne blant annet ha bedret Mobitras. Jeg vil av den grunn konkludere med at et rammeverk bør inneholde en generell struktur for å kunne behandle flere ulike typer eksterne sensorsystemer.

Interne sensorsystemer av interesse

Mange av eksempelapplikasjonene er oppmerksomme på brukerens tid. Mobitras benytter for eksempel tilbakelagt treningstid for å avgjøre om en utøver har passert en skyggeannotering. I The Conference assistant blir i tillegg dato anvendt for å avgjøre hvilken forelesning brukeren befinner seg på. Både tid, dato, og hans lokasjon er nødvendig for å bestemme dette. Jeg har tidligere definert tid og dato til å være brukerens tidskontekst (jamfør 3.1.1). Brown argumenterer for at man kan ha behov for å assosiere informasjon til ulike tilstander som kan inntreffe internt i maskinen. Et eksempel på dette kan være dersom temperaturen på prosessoren skulle overstige en maksimumsverdi, eller forbindelsen til nettverket skulle endre seg vesentlig [Brown 1995]. Det siste benyttes i The Lancaster Guide. Om nettverksforbindelsen skulle falle bort der, blir applikasjonen satt i en egen *off-line* modus. Da blir kun lokalt oppbevart informasjon tatt i bruk. Som nevnt er nettverket her en del av lokasjonssystemet, slik at om dette faller bort mister applikasjonen også oversikt over brukerens posisjon. Nettverksforbindelsen har jeg tidligere definert som en del av brukerens systemkontekst (jamfør 3.1.1).

Blant eksempelapplikasjonene blir ikke interne sensorsystemer benyttet til annen registrering enn den nevnt over. Sannsynligvis medfører dette at mange av de interne sensorløsningene også kan konstrueres og implementeres som en del av rammeverket.

Dynamisk valg av sensorløsninger i forhold til tjenestekvalitet

> *"Straks Kari nærmer seg kjøpesenteret dukker følgende melding opp fra Mocado: 'Vinterklær kan kjøpes hos sportsforretningen i annen etasje'. Kari bærer ikke en termometersensor selv, men derimot henger det et slikt på utsiden av butikkvinduet. Denne har Mocado tatt i bruk ved hjelp av trådløs kommunikasjon, og funnet at temperaturen har sneket seg under -5°C. Straks hun kommer inn i butikken kan imidlertid ikke termometerets målinger vurderes som gyldige: De gjaldt bare utendørs. Sensoren kobles derfor fra."*

Chen hevder at dersom mobilitetsbaserte kontekstoppmerksomme applikasjoner skal være hensiktsmessige å anvende i praksis, bør brukeren kunne ta med og legge igjen sensorer som det passer [Chen et al. 2000]. Om han for eksempel ikke ønsker å benytte pulsmåleren en dag, skal han slippe det. I et slikt tilfelle må applikasjonen være forberedt på at sensorer kan kobles til og fra midt under eksekvering. I eksempelet med Kari ble et slikt tilfelle demonstrert. Her valgte Mocado selv å benytte en sensor den fant i nærheten av brukeren[1]. Det spesielle jeg ønsker å trekke frem fra dette eksempelet, er hvordan denne sensoren bare produserte gyldige observasjoner for dem som befinner seg på utsiden av kjøpesenteret. Straks brukerne kommer inn, kan den ikke lenger si noe om deres omgivelser. Termometeret leverer med andre ord bare gyldig informasjon for bestemte situasjoner. Dette tilfellet kaller Pascoe for *contextual resource discovery* [Pascoe 1998]. Mer spesifikt innebærer dette at applikasjonen tar i bruk kontekstoppmerksomhet for å lete frem tilgjengelige ressurser som kan anvendes av PDA-en (som i dette tilfellet var en sensor). Pascoe hevder at slike ressurser bare kan anvendes dersom brukeren (eller PDA-en) befinner seg innenfor samme kontekstuelle situasjon som ressursene. I tilfellet over ble for eksempel termometeret benyttet fordi det befant seg på noenlunde samme lokasjon som Kari.

Ofte kan flere sensorer levere samme informasjon av samme type. I Cyberguide blir for eksempel to forskjellige eksterne posisjoneringssystemer benyttet. Hvilket som skal velges, avhenger av brukerens situasjon. Er han ute skal GPS benyttes, er han inne skal valget falle på 3D-iD Pinpoint[2]. Applikasjonen er selv i stand til å bytte mellom disse under eksekvering. Det som er avgjørende for applikasjonens valg, er sensorenes tjenestekvalitet; Mens GPS er avgrenset til å bare avgi gyldig informasjon ute, kan 3D-iD Pinpoint kun anvendes inne.

---

[1] Dette er bare et tenkt eksempel. Mocado er ikke blitt implementert slik i virkeligheten.

[2] Mer om denne teknologien kan leses hos http://www.rftechnologies.com/ (dato: 29.07.2002).

Dey benevner mengden av alle forskjellige målinger en sensor kan levere som sensorens *dekning* [Dey 2000]. Dekning er med andre ord et mål på tjenestekvalitet. Denne måleenheten gjelder ikke bare avgrensninger innenfor fysisk lokasjon, men også alle andre parametere som er med på å begrense sensorens bruksområde. Et termometer som leverer målinger mellom 0°C til 10°C kan for eksempel få et dekningsproblem under vinterlige forhold når det er minusgrader. Dey hevder at det også finnes fem andre skalaer som beskriver sensorers tjenestekvalitet; *nøyaktighet*, *stabilitet*, *oppløsning*, *oppdateringsfrekvens*, og *tilbakemeldingstid*[1]. Kort beskrevet går disse ut på [Dey 2000]:

Nøyaktighet – definerer hvor nøyaktig sensorens målinger er. Mens noen lokasjonssystemer kan bestemme brukerens posisjon med en nøyaktighet på +/- 5 meter, kan for eksempel andre ha en nøyaktighet på bare noen millimeter.

Stabilitet – definerer hvor stabil en sensor er i forhold til å skape feilsituasjoner. Dey hevder at utviklere normalt sett vil kreve at en applikasjon skal være i stand til å takle ustabile sensorer (uten at applikasjonen terminerer av den grunn).

Oppløsning – definerer hvor store endringer som må inntreffe i sensorens domene før en oppdatering blir levert. Et termometer kan for eksempel spesifisere at den bare leverer oppdateringer når temperaturen har endret seg en hel grad.

Oppdateringsfrekvens – definerer hvor ofte en sensor kan levere oppdateringer. En GPS-mottaker kan for eksempel spesifisere at den leverer oppdateringer hvert andre sekund.

Reaksjonstid – definerer tiden det tar fra en hendelse inntreffer til sensoren leverer en oppdatering. Et alarmanlegg kan for eksempel bruke lang tid på å registrere at en innbruddstyv har brutt seg inn i et hus. Dette vil igjen medføre lang tid i forhold til en situasjonsoppdatering i applikasjonen. Dey hevder at utviklere aller helst ser at denne verdien er tilnærmet lik null for de fleste sensorer.

Som nevnt bør brukeren delta minst mulig når det gjelder spesifikasjon av kontekstuell informasjon. Dette inkluderer også hvilke sensorer som skal anvendes. Slik som Cyberguide demonstrerer, bør dette skje automatisk. Jeg vil hevde at det er minst to spørsmål som må kunne besvares om et rammeverk skal være i stand til å foreta slike valg:

1. Leverer sensoren som skal benyttes gyldige observasjoner i forhold til brukerens kontekst? (sensorens dekning).
2. Om flere sensorer leverer gyldige observasjoner, hvilke passer da best med tanke på 1) applikasjonens krav, og 2) sensorenes tjenestekvalitet?

Hvilke sensorer som passer best med tanke på tjenestekvalitet, bør kunne varieres i henhold til hvilke parametere som er viktigst for sensortypen. Lokasjonssystem-

[1] På engelsk benevnes disse som henholdsvis *coverage*, *reliability, resolution, frequency* og *timeliness* [Dey 2000].

er kan for eksempel evalueres etter nøyaktighet, mens klokker kan rangeres etter oppløsning.

Det er naturlig å tenke seg at sensorer kan flytte på seg rent fysisk. I så fall vil dette kunne medføre endringer i forhold til dens tjenestekvalitet. Et termometer som hang fast på en bil ville endre dekningsområde i takt med bilens posisjon. Dette gjør det vanskelig for slike sensorer å garantere en viss tjenestekvalitet på forhånd.

7.1.2 Kommunikasjon mellom applikasjoner

Kontekst kan også registreres ved hjelp av nettverkskommunikasjon. I The Lancaster Guide blir for eksempel et trådløst nettverk anvendt for utveksling av lokasjon. Fordi applikasjonene kjenner sin egen brukers lokasjon, blir dette distribuert og offentliggjort over nettverket. På den måten får alle turistene informasjon om hvor andre i reiseføget befinner seg.

Andres nærhet, identitet, aktivitet, interesser, og lignende, benevnes av Salber som sosial kontekst [Salber et al. 1998]. For det meste er det også slik informasjon som blir registrert ved hjelp av kommunikasjon. Ved å distribuere brukernes kontekst mellom applikasjonene, blir mye sosial kontekst avdekket. I tillegg til The Lancaster Guide benytter også The Conference assistant trådløs kommunikasjon. Her er dessuten ikke lokasjon alene om å bli overført; opplysninger om besøkte forelesninger, notater gjort under disse, andre aktiviteter brukerne planlegger å overvære, og lignende blir overført regelmessig. Dette blir blant annet anvendt for å informere brukerne om aktiviteter de bør vurdere å overvære. Om et flertall av kollegene til en bruker har planlagt å delta på en forelesning, er sannsynligheten stor for at denne også kan være relevant for han.

Jeg vurderte å implementere bruk av trådløs kommunikasjon i Mocado. Målet var det samme som over; oppnå sosial kontekst i form av andres posisjon. I tillegg kunne en slik kommunikasjon tillate utveksling av annoteringer. Jeg forsøkte å realisere dette ved hjelp av teknologien BlueTooth[1], men da dette ikke så ut til å virke etter et initsielt forsøk valgte jeg heller å sette i gang med Mobitras. Likevel vurderer jeg kommunikasjon, ut ifra det ovenstående, som en viktig kilde for å registrere brukerens kontekstuelle situasjon.

7.1.3 Manuell spesifikasjon og overstyring

Som nevnt er det ønskelig å registrere mest mulig av en brukers kontekst automatisk [Lieberman et al. 2000, Dey 2000]. Likevel er det ikke alltid dette lar seg så lett gjøre. Hva en bruker tenker eller interesserer seg for vil for eksempel være vanskelig å avsløre ved hjelp av sensorer alene. Blant eksempelapplikasjonene blir slik informasjon isteden oppført av brukeren manuelt.

---

[1] Mer om denne kan leses hos ”http://www.bluetooth.com” (dato: 29.07.2002).

Hva som vurderes som manuell kontekstspesifikasjon og hva som er vanlig interaksjon med applikasjonen, kan flyte litt over i hverandre. For eksempel kan en avtale i brukerens kalender benyttes for å registrere hans aktivitetskontekst. Opprettelse av en slik avtale er imidlertid nærmere det jeg vil vurdere som normal interaksjon med PDA-en, fremfor kontekstspesifikasjon. Likevel vil jeg benevne all informasjon som har sin opprinnelse fra bruker-maskin-interaksjon som manuell kontekstspesifikasjon. Jeg har i så tilfelle identifisert tre kontekstkategorier hvor brukeren deltar aktivt. Disse kategoriene er: *Identitet*, *aktivitet*, og *interesser og preferanser*[1].

Identitet – er kontekst som omhandler en brukers personalia. Stort sett blir dette anvendt for å identifisere nære personer (sosial kontekst). Med andre ord er det ikke brukerens egen identitet som er interessant, men den til de i nærheten. I The Lancaster Guide kan for eksempel turistene få frem informasjon om hvor andre i reiseføLget befinner seg. For at dette skal la seg gjennomføre, må hver enkelt turist oppgi personalia før de tar applikasjonen i bruk. Det samme er tilfellet i The Conference assistant; for at deltagerne skal finne hverandre igjen på konferansen, må personalia oppgis i applikasjonen.

Aktivitet – spesifiserer hva slags oppgaver brukeren har holdt på med, både i fortid og nåtid. Ofte kan mye av dette registreres fra hans elektroniske kalender, om en slik ligger tilgjengelig. I The Conference assistant blir for eksempel deltagerne informert når en oppført aktivitet er i ferd med å begynne.

Interesser og preferanser – beskriver hva slags informasjon brukeren ønsker å motta. I The Conference assistant er slik informasjon med på å avgjøre hva slags aktivitetsforslag de får presentert under messen. Viser det seg at en deltager bedriver forskning innen et spesielt fagfelt, blir forelesninger rundt dette prioritert. Også The Lancaster Guide benytter slik informasjon. Turister som får presentert spesielt tilpassede turer rundt i byen, får disse basert på sine oppførte interesser og preferanser. For begge applikasjonene er brukerne nødt til å legge inn interessekontekst manuelt i applikasjonene på forhånd.

Kontekst som tilhører kategoriene over er gjerne vanskelig å registrere uten hjelp fra brukeren. Jeg mener at grunnen til dette kommer av at slik informasjon vanskelig lar seg måle. Uavhengig av brukeren, kan en applikasjon måle seg frem til både hans posisjon og hjerterytme, men hva slags interesser han har kan strengt tatt bare avgjøres av han selv. Likevel kan noe av dette utledes fra andre deler av hans kontekst. Står han lenge nok på en bussholdeplass, kan applikasjonen med relativt stor sikkerhet anta at han venter på bussen. Imidlertid kan det også hende at han står der fordi han har truffet noen kjente, eller at han følger med på en gatesjonglør som opptrer ved siden av. I så fall blir dette spekulasjoner og av den grunn lite egnet informasjon om man ønsker å si noe sikkert om hans situasjon. Brown hevder at det vil være ugunstig om en bruker til stadighet skulle få presentert irrelevant informasjon, da dette vil oppleves som irriterende [Brown 1995]. Om slikt ikke skal skje, er man avhengig av å tolke hans kontekst riktig.

---

[1] I 3.1.1 definerte jeg kategorien ”interesser og preferanser” som en underkategori av ”identitet”.

Oppsummert vil jeg hevde at enkelte deler av en brukers kontekst vanskelig lar seg måle ved hjelp av sensorer. Jeg vil derfor konkludere med at et rammeverk bør inkludere god støtte for manuell kontekstspesifikasjon.

7.1.4 Historikk

En fjerde metode for å registrere kontekst er bruk av historikk. Kan en situasjon i fortid være med på å forklare en situasjon i dag? Vurderer man spørsmålet ut ifra eksempelapplikasjonene er svaret "ja". Long hevder følgende:

> *"As the prototypes of Cyberguide evolve, we will be able to handle more of the user's Context, such as where she has been, what was seen and heard there, as well as where others are and have been."*
>
> – [Long et al. 1996, side 1].

Bruk av historisk kontekst kommer til syne i Mobitras. For å kunne gjennomføre en trening, er applikasjonen tvunget til å holde rede på utøverens passerte sekunderingspunkter underveis. Dette blir anvendt under aktiveringsprosessen. I tillegg blir historikk benyttet for å konstruere skygger. Likevel vurderer jeg ikke skyggene som et eksempel på historisk kontekst. Dette begrepet er forbeholdt informasjon som forklarer brukerens situasjon i nåtid. De er heller et eksempel på nære (virtuelle) personer, og derfor sosial kontekst. Uansett vurderer jeg skyggene i seg selv som veldig applikasjonsspesifikke for Mobitras, og de vil av den grunn heller ikke avsløre noen generelle krav for et rammeverk.

I Kapittel 5.4 (om Mocado) identifiserte jeg et lignende behov som Mobitras når det gjelder aktiveringshistorikk. Her ønsket jeg meg en mulighet for å endre på informasjonen som ble presentert når en annotering ble aktivert for andre eller tredje gang. Noe av det samme er også et tema i Cyberguide. Her benytter de posisjonshistorie kombinert med tid for å avgjøre om kontekstrelevant informasjon er blitt gyldig. Denne informasjonen bestemmer brukerens synsretning. Om en besøkende har observert en attraksjon eller ikke, er imidlertid ikke bare avhengig av synsretning. En aktivering kan for eksempel inntreffe om man streifer forbi en attraksjon tilfeldig. En vesentlig opplysning er *hvor lenge* man har stått og sett på den. En lang besøkstid indikerer en stor sannsynlighet for at informasjonen er blitt lest. I ettertid ser jeg at det samme forholdet også burde gjelde for Mocado og dens annoteringer.

En mer direkte bruk av historisk kontekst blir også benyttet. Pascoe forklarer at en historie over sensorenes data i The Field assistant blir lagret i tilfelle noen av sensorene skulle slutte å levere observasjoner for kortere perioder [Pascoe 1998]. I så tilfelle blir den oppbevarte historien anvendt for å simulere hvilke nye data som sannsynligvis *skulle* ha kommet. Sagt på en annen måte: En historie over tidligere registrert kontekst blir benyttet for å avgjøre brukerens situasjon i nåtid.

En lignende anvendelse av historikk som hos The Field assistant vises også i The Conference assistant. Her har man mulighet (via en støtteapplikasjon) å i ettertid hente frem en del av den konteksten som inntraff på en konferanse i fortid. Man kan for eksempel se når man ankom eller forlot de forskjellige aktivitetene, hvilke spørsmål som ble stilt underveis, og hvor kollegene var på forskjellige tidspunkter. Her blir med andre ord konteksthistorien presentert som informasjon direkte til brukeren. Likevel vurderer jeg ikke dette som historisk kontekst. Grunnen er at den ikke benyttes av applikasjonen som informasjon for å forklare hans situasjon i nåtid. Imidlertid belyser dette likevel at en oppbevaring av registrert kontekst er et reelt behov blant flere av oppgavens applikasjoner.

7.1.5 Kontekstutledning

Som beskrevet tidligere blir brukerens posisjon vurdert som den desidert viktigste kontekstuelle parameteren for mobilitetsbaserte applikasjoner (jamfør 3.4). En del hevder at den også er en slags nøkkelparameter [Dix et al. 2000, Chen et al. 2000]. Denne oppfatningen gjenspeiles også i eksempelapplikasjonene; alle gjør bruk av posisjon.

Det viser seg at man kan finne ut relativt mye om en brukers kontekstuelle situasjon om man kjenner hans geografiske posisjon. Man kan for eksempel oppspore hvilken gate han står i, eller kanskje hvilken by? Kan man bestemme hvilket land, bør også språket la seg avgjøre. Kombinerer man dette med annen kjent kontekst, blir mulighetene enda flere. I The Conference assistant blir for eksempel lokasjon, tid, og dato kombinert for å finne ut hvilken forelesninger deltagerne befinner seg på (se Figur 25). I Cyberguide blir brukerens posisjoner over tid analysert til å avgjøre hvilken retning de står orientert (hva de ser på). Applikasjonene avdekket med andre ord andre deler av brukerens kontekst ved å analysere den delen som allerede var kjent. Dey hevder at applikasjoner har mye å hente på dette, da metoden utvinner kontekst på et høyere, ikke-sansbart nivå [Dey 2000]. Dette blir av han vurdert som selve nøkkelen for å produsere mer sofistikerte kontekstoppmerksomme applikasjoner.

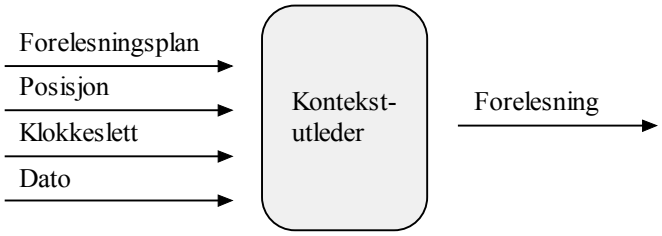
Fremgangsmåten over vil i denne oppgaven benevnes som *kontekstutledning*[1]. Informasjonen som blir analysert vil kalles *prekontekst*, og informasjonen som utledes for *postkontekst*. Innenfor dette området er det spesielt to forhold som bør belyses. Det ene gjelder forholdet mellom kontekstutledning og kontekstomforming, og det andre er utledning som en sirkulær prosess.

Kontekstomforming

En rekke applikasjoner har benyttet det jeg vil benevne som kontekstomforming; det å endre presentasjonsformatet på kontekstuell informasjon fra ett format til et annet. For eksempel kan antall minutter (prekontekst) omformes til antall sekun-

[1] Dette er også blitt benevnt som ”Contextinterpretation” [Dey et al. 1998, Dey 2000].

der (postkontekst). Informasjonen er den samme, men formatet er annerledes. Hvor mange sekunder en utøver har tilbakelagt i Mobitras er like mye en del av hans kontekst som antall minutter. Kontekstomforming vurderer jeg imidlertid for å være en form for kontekstutledning; også her blir kjent kontekstuell informasjon analysert for å utvinne mer kontekstuell informasjon. Forskjellen er at man i kontekstutledning også produserer en annen type informasjon, ikke den samme informasjonen bare på et annet format.



Figur 25: Kontekstutledning – Figuren viser hvordan kjente deler av brukerens kontekst kan benyttes til å utlede hvilken forelesning han befinner seg på. Her er blant annet posisjon prekontekst, mens forelesning er postkontekst.

I CyberDesk[1] benyttes noen komponenter benevnt *type converters* til kontekstomforming [Dey et al. 1998]. Disse mottar én type kontekst som input, og leverer tilbake en annen som er blitt omformet. Dey beskriver selv et eksempel på bruk av dette [Dey et al. 1998]: CyberDesk inneholder en komponent som undersøker alle tekster brukeren skulle finne på å markere i en eller annen applikasjon (applikasjonskontekst, jamfør 3.1.1). Om disse tekstene kan omformes til en e-post adresse, blir denne adressen gjort tilgjengelig for alle de andre applikasjonene tilknyttet systemet. Dette kan for eksempel være tilfellet om den markerte teksten var et navn. En e-postapplikasjon kan dermed plukke opp den identifiserte adressen, og tilby brukeren å sende et brev til den identifiserte personen. Dette er med andre ord funksjonalitet som skapes ut ifra kontekstoppmerksomhet [Dey et al. 1998].

Både i Mocado og Mobitras benyttet jeg kontekstomforming for å endre dataene jeg mottok fra GPS-mottakeren. Disse ble parsert og strukturert fra én tekststreng til flere tallverdier (koordinater). Med disse koordinatene ble det først og fremst enklere å behandle kart og lokasjon. Også i Field assistant observerer jeg det samme behovet. Her benytter Pascoe det han kaller *synthesizers* [Pascoe 1998]. Disse har akkurat den samme oppgaven som beskrevet over; omforme kontekst fra et format til et annet.

[1] CyberDesk er en kontekstoppmerksom applikasjon utviklet for å integrere uavhengige applikasjoner tettere sammen. Ved å la slike applikasjoner kommunisere hva slags tjenester de har å tilby, kan brukeren lettere anvende funksjonalitet utover hva en spesiell applikasjon kan tilby alene. CyberDesk er utviklet for stasjonær bruk, og er derfor ikke mobilitetsbasert.

Brown, med flere, vurderer opplysninger om nære objekter som interessant kontekstuell informasjon [Brown 1995, Dey et al. 1998]. Jeg har identifisert to ulike metoder for å fremskaffe slike opplysninge:. Enten ved hjelp av kommunikasjon mellom entiteter (jamfør 7.1.2), eller ved hjelp av kontekstutledning. I det første tilfellet kan gjenstander (som for eksempel printere) avsløre sin egen lokasjon på lik linje med hvordan The Lancaster Guide distribuerer lokasjon; ved hjelp av nettverkskommunikasjon. I det andre tilfellet kan kontekstutledning anvendes. Ved å benytte et "kart" over objektenes lokasjon som kilde, sammen med brukerens lokasjon som søkekriterium, kan man foreta søk i kartverket for å finne eventuelle nære objekter.

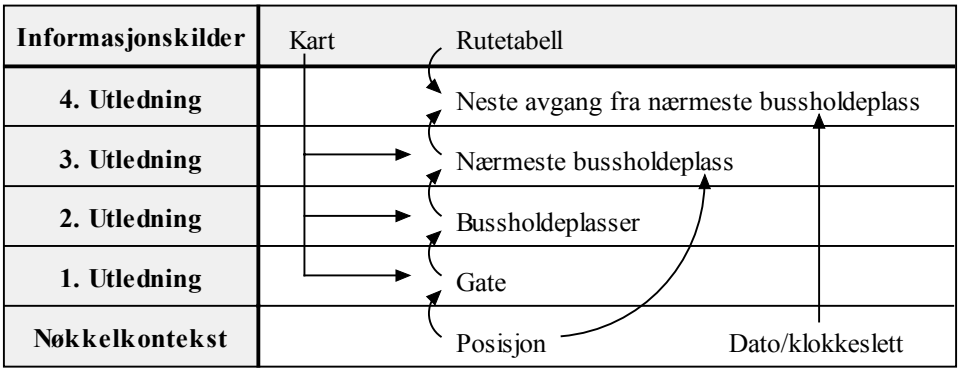
Oppsummert vurderer jeg kontekstutledning (og kontekstomforming) som en viktig teknikk for å registrere brukerens kontekstuelle situasjon.

Utledning som en sirkulær prosess

Kontekstutledning kan også foregå i flere runder. I CyberDesk [Dey et al. 1998] demonstreres dette. Her kan uavhengige utledningskomponenter (*type converters*) lese tilgjengelige deler av brukerens kontekst, og legge tilbake utledet postkontekst. Siden den utledede konteksten blir tilgjengelig for andre utledningskomponenter i systemet, tillater dette disse å fortsette utviklingen videre. Slik kan kontekstutvikling i CyberDesk pågå i en sirkulær prosess.

Et eksempel på repeterende kontekstutvikling illustreres i Figur 26. Her blir et knippe nøkkelkontekst utledet i flere runder. For hver utledningsrunde som gjennomføres, blir nye deler av konteksten avdekket. Til sist, i runde fem, har systemet klarlagt når neste buss forlater nærmeste bussholdeplass, sett i forhold til utgangsposisjonen. Jeg vil presisere at det ikke alltid er om å gjøre å komme til toppen av et slikt hierarki. Om dette eksempelet var hentet fra CyberDesk, så kunne det like gjerne hende at store deler av den utledede informasjon ikke ble anvendt til noe som helst. Nytteverdien avgjøres isteden av applikasjonene som er tilknyttet systemet.

Konteksten nederst i et utledningshierarki har jeg benevnt som nøkkelkontekst (jamfør 3.1.1, og 3.4). Dette er kontekstuell informasjon som fungerer som indeksparametere inn i andre informasjonskilder [Chen et al. 2000]. Chen hevder at spesielt lokasjon, tid og dato er en del av denne. Bruk av kontekstutledning i eksempelet i Figur 26 illustrerte også dette i praksis; både lokasjon og tidspunkt fungerte her som indeksparametere for å avgjøre når neste buss skulle ankomme på nærmeste holdeplass. Informasjonskildene her var kart og rutetabell.



Figur 26: Kontekstutledning – Figuren viser et eksempel på hvordan kontekst kan utledes fra kontekst i flere omganger.

7.2 Anvendelse i forhold til brukerne

Over har jeg studert hva slags teknikker som ble anvendt blant eksempelapplikasjonene for å registrere en brukers kontekst. Samtidig drøftet jeg også hva slags informasjon de ulike metodene utvant. Slik jeg ser det, er disse forholdene i utgangspunktet ikke så interessante for applikasjonsutviklerne. Hvordan applikasjonene registrerer kontekst bør isteden gjøres transparente i forhold til dem. Av den grunn ville det også passe bra om dette kunne implementeres vekk i et rammeverk.

Den andre delen jeg vil belyse i dette kapitlet, er anvendelse av kontekstoppmerksomhet i forhold til brukerne. Dette er funksjonalitet jeg vurderer som relevante i forhold til hvilke behov *de* har. Og funksjonalitet som er relevant i forhold til brukernes behov bør gjenspeiles i applikasjonene, og videre også i et rammeverk for slike. Av den grunn blir denne analysen også relevant for oppgaven.

Forholdene jeg har funnet relevante for brukerne er: Situasjonssimulering, presentasjon av kontekst, og private og offentlige aspekter rundt kontekstutveksling.

7.2.1 Situasjonssimulering

I Kapittel 7.1.3 drøftet jeg behovet for manuell kontekstspesifikasjon. Hovedtrekkene var at man i utgangspunktet ønsket registrering av kontekstuell informasjon automatisk, men at enkelte deler likevel lot seg best beskrive av brukeren manuelt. Brown snur litt på argumentene for automatisk registrering, og hevder at manuell kontekstspesifikasjon også kan være fordelaktig som en egen funksjon i applikasjonene [Brown 1995]. Brown hevder følgende:

> *"Clearly it is a limitation if the only way a tourist can see information about a place is to actually go there (not for-*

> *getting to take a GPS receiver attached to their PDA). It is much better if the user can, as an alternative, pretend to be at the place, and thus trigger the information: they can then judge whether the place is worth visiting. In the user interface pretence can be implemented by allowing the user to point at a map, thus pretending they are at the location pointed at, and temporarily overriding the real setting for their location as given by a sensor."*
>
> – [Brown 1998b, side 4].

Kontekstoverstyring var også noe jeg fant hendig i Mocado; ved å manuelt bestemme parametere som posisjon, kunne jeg plassere meg akkurat der jeg selv ville i applikasjonen. Dermed kunne jeg simulere at jeg var helt andre steder en hva som faktisk var tilfellet. Dette var behagelig om jeg ønsket å få frem informasjon om steder jeg hadde besøkt tidligere, eller planla å besøke senere. Mye av den samme tankegangen er også implementert i Field Assistant. Når man legger inn en ny observasjon der, blir lokasjon og tidspunkt automatisk fylt inn som observasjonens kontekst. Likevel har man full mulighet til å overstyre disse verdiene om ønskelig. Dermed kan man også legge inn observasjoner for andre steder enn der han faktisk er, eller til andre tider. Pascoe hevder dette er viktig om feltarbeiderne skal kunne bevege seg samtidig som de legger inn nye observasjoner.

Som en oppsummering vil jeg hevde at man alltid bør ha muligheten til å overstyre brukerens kontekst i kontekstoppmerksomme applikasjoner.

7.2.2 Kontekst som kontekstrelevant informasjon

I Kapittel 3.2 beskrev jeg noen ulike anvendelsesområder for kontekstoppmerksomhet. I denne oppgaven har jeg valgt å rette søkelyset på en av dem, nemlig behandling av kontekstrelevant informasjon. Pascoe argumenterer for at dette er assosiering av informasjon til kontekst, og benevner kategorien som *contextual augmentation* [Pascoe 1998]. I tillegg definerer han kategorien *contextual sensing*. Dette er når applikasjonen anvender kontekst for å informere brukeren om hans situasjon direkte (jamfør 3.2). Etter å ha studert oppgavens område mer grundig, vurderer jeg disse kategoriene som til dels overlappende. For å illustrere dette, kan man se for seg et eksempel med en kontekstoppmerksom applikasjon. Denne er i stand til å finne nærmeste bussholdeplass i forhold til brukeren, målt i avstand. Spørsmålet er så: Er informasjon om nærmeste bussholdeplass kontekstuell informasjon, eller kontekstrelevant informasjon? På den ene siden fant jo applikasjonen frem til denne informasjonen ut ifra kjennskap om brukerens lokasjon. Slik sett er informasjonen kontekstrelevant. På den andre siden kan jo informasjon omkring nære steder vurderes som en del av brukerens kontekst (jamfør 3.1.1). Svaret avhenger heller, slik jeg ser det, av hvordan informasjonen anvendes. Både kontekstuell-, og kontekstrelevant informasjon kan ses på som enten

kontekstuell-, og kontekstrelevant informasjon. Sagt på en annen måte; presentasjon av kontekstuell informasjon kan i mange tilfeller vurderes som presentasjon av kontekstrelevant informasjon.

Blant eksempelapplikasjonene blir presentasjon av kontekst anvendt mye. Ikke minst gjelder dette posisjon. Stort sett alle applikasjonene fremviser denne parameteren i ett eller annet format. Både i Mocado, The Field assistant, og Cyberguide får man fremvist sin geografiske posisjon på et kart. I The Lancaster Guide får man i tillegg frem hvor andre turister befinner seg. Dette er presentasjon av sosial kontekst (jamfør 3.1.1).

I Mobitras er presentasjon av posisjon utelatt. Imidlertid får utøveren lest opp mye annen kontekst, først og fremst sekunderingsinformasjon. Både tid, tilbakelagt avstand, avstand til skygger, og lignende, er presentasjon av kontekst. Man kan også vurdere skyggepasseringer er en del av dette, siden slike passeringer også beskriver utøverens situasjon. Imidlertid er slik informasjon realisert som annoteringer, og derfor kontekstrelevant informasjon. Dette er et annet eksempel på hvordan kontekstuell-, og kontekstrelevant informasjon kan vurderes som samme abstraksjon.

Jeg har tidligere argumentert for at man bør ha fri tilgang til å overskrive, eller bestemme, brukerens egen kontekst. I tillegg vil jeg nå hevde at man også bør ha fri tilgang til å lese den. Dette vil blant annet tillate direkte presentasjon av kontekstuell informasjon for brukeren.

7.2.3 Privat og offentlig anvendelse av kontekst

Som nevnt ble en brukers personlige kontekst distribuert rundt og offentliggjort i flere av eksempelapplikasjonene. Dette var en grei måte å avsløre sosial kontekst. Spørsmålet jeg likevel stiller er om fri distribusjon av kontekst kan frarøve enkeltmenneskers ønske om å beskytte privatlivet. Chen argumenterer for at folk flest ikke liker tanken på at noen til en hver tid kan se hvor man befinner seg [Chen et al. 2001]. Om slike data i tillegg blir loggført, vokser misnøyen ytterligere. Chen hevder videre:

> *"Perfect privacy guaranties are in general hard (and expensive) to provide. The user should be able to have control over their contextual information and over who may gain access to it. The system architecture needs to provide user-controllable tradeoffs between privacy and both functionality and efficiency. But it is difficult to be specific about what context information should be visible to who, and when."*
>
> – [Chen et al. 2001, side 12].

Muligheten for å verne om privatlivet er heller beskjedent representert blant eksempelapplikasjonene. I The Conference assistant kan deltagerne få frem kontekstuell informasjon omkring sine egne kolleger. Om applikasjonen likevel har avgrenset denne muligheten til bare å omhandle forhåndsbestemte grupper, er uvisst (jeg fant ikke noe detaljert informasjon om dette). Det samme er også tilfellet i The Lancaster Guide; her mener jeg likevel å ha forstått det slik at man kan få frem lokasjonen til de man eventuelt kjenner navnet på.

Det at informasjon omkring beskyttelse av privatliv er såpass beskjedent kommentert innen litteraturen jeg anvender, forteller kanskje noe om vektleggingen av dette hos utviklerne. Muligens er dette nedprioritert. Likevel tror jeg ikke slike forhold blir vurdert som irrelevante. Fokus er isteden lagt til andre deler av applikasjonene. Bare det å benytte kontekstoppmerksomhet blir i seg selv vurdert som problematisk [Dey 2000]. Tross alt er alle eksempelapplikasjonene prototyper. Dette er også tilfellet med Mocado og Mobitras. Selv om jeg og mine veiledere har hatt flere diskusjoner omkring sikkerhet underveis, vurderer jeg likevel dette til å være et problem som ville gjøre seg gjeldene etter hvert. Jeg har frem til nå hatt mer enn nok med å få applikasjonene til å fungere uten slike restriksjoner. Imidlertid mener jeg at et rammeverk bør være i stand til å behandle slike spørsmål utfyllende.

7.3 Oppsummering

Jeg valgte å dele dette kapitlet inn i to deler; anvendelse av kontekstoppmerksomhet i forhold til applikasjonene, og i forhold til brukerne. Jeg innledet videre den første delen med å introdusere to viktige spørsmål: Hva slags kontekstuell informasjon gjør kontekstoppmerksomme applikasjoner bruk av, og hvordan registrerer de den? Ved å analysere eksempelapplikasjonene kom jeg frem til seks forskjellige metoder for registrering av kontekst. Disse, sammen med den kontekstuelle informasjonen de registrerte, kan oppsummeres som følger:

<u>Eksterne sensorløsninger</u>

> *Beskrivelse*: Eksterne sensorsystemer registrerer kontekstuell informasjon ved hjelp av eksternt tilkoblet utstyr. Denne metoden registrerer mye nøkkelkontekst, først og fremst lokasjon. Spesielt viktig var også muligheten for å kunne skifte mellom ulike sensorer. Dette burde helst foregå automatisk, og etter følgende tjenestekvalitetskrav: dekning, nøyaktighet, stabilitet, oppløsning, oppdateringsfrekvens, og reaksjonstid.
>
> *Anvender*: GPS, trådløst nettverk, termometer, pulsmåler, og lignende. Mange ulike innendørs lokasjonssystemer ble benyttet. Utendørs var GPS oftest representert.
>
> *Registrerer*: Fysisk kontekst: Lokasjon, temperatur, puls, lyd, og lignende.

<u>Interne sensorløsninger</u>

Beskrivelse: Interne sensorløsninger registrerer kontekst uten bruk av eksternt tilkoblet utstyr.

Anvender: Tilgang til informasjon fra operativsystemet.

Registrerer: Tidskontekst: Tid og dato. Systemkontekst: Båndbredde, prosessorkraft, minnekapasitet, og lignende.

Kommunikasjon mellom applikasjoner

Beskrivelse: Kontekst registreres ved hjelp av informasjonsutveksling mellom applikasjoner.

Anvender: Trådløst nettverk.

Registrerer: Fysisk kontekst: Nære gjenstander. Sosial kontekst: Andres lokasjon, interesser, preferanser, identitet, aktivitet, emosjonelle tilstand, og lignende.

Manuell spesifikasjon og overstyring

Beskrivelse: Kontekst registreres ved at brukeren eksplisitt spesifiserer kontekstuell informasjon til applikasjonen. Spesielt var dette viktig for å kunne tillate situasjonssimulering.

Anvender: Ingenting.

Registrerer: I utgangspunktet alle kategorier, men hovedsaklig aktivitet, interesser, preferanser, og identitet.

Historikk

Beskrivelse: Kontekst registreres ved å benytte opplysninger om situasjoner inntruffet i fortid.

Anvender: Tilgang til logg over tidligere registrert kontekstuell informasjon.

Registrerer: Historisk kontekst: Aktiveringshistorikk, lokasjonshistorikk, og ellers annen kontekshistorikk som benyttes for å beskrive brukerens situasjon.

Repeterende kontekstutledning

Beskrivelse: Kontekstutledning består av å registrere kontekst ved hjelp av å analysere, omforme, og kombinere kjente deler av brukerens kontekst. Kan gjøres enda kraftigere ved å tillate repeterende kontekstutledning. En enklere form kalt kontekstomforming ble også identifisert som et spesialtilfelle av kontekstutledning.

Anvender: Eksterne informasjonskilder og allerede kjent kontekst.

Registrerer: I utgangspunktet alle kategorier. Gjerne høyere, ikke-sansbar, kontekst, eller allerede kjent kontekst, men i andre formater.

I kapitlets andre del gjorde jeg rede for hvordan kontekstoppmerksomhet ble tilgjengeliggjort for brukerne. I utgangspunktet ønsket jeg at dette skulle foregå uten at de var nødt til å delta. Noe av målet med kontekstoppmerksomhet var, som nevnt, å minske brukerinteraksjonen [Dey 2000]. Likevel var det spesielt tre forhold som ga en indikasjon også på det motsatte. Det ene var muligheten for situasjonssimulering. Dette var en ekstrem form for manuell kontekstspesifikasjon der brukerne kunne plassere seg selv i andre tenkte situasjoner (og steder) enn den de egentlig var i. Dette lot seg gjøre ved å overstyre konteksten som applikasjonene benyttet.

Jeg argumenterte også for likheten mellom kontekstuell-, og kontekstrelevant informasjon. Ofte kunne presentasjon av kontekstuell informasjon også vurderes som presentasjon av kontekstrelevant informasjon. Det viste seg også at eksempelapplikasjonene presenterte kontekstuell informasjon for brukeren ofte. Spesielt gjaldt dette lokasjon og sosial kontekst: De fleste tegnet opp brukerens posisjon på et kart.

Helt til sist drøftet jeg forholdet mellom privat og offentlig anvendelse. Spesielt registreringsmetode nummer tre, kommunikasjon, benyttet kontekstutveksling som en kilde til sosial kontekst. Likevel kan det være mindre ønskelig å distribuere private opplysninger om seg selv uten forbehold. Jeg drøftet også i hvilken utstrekning dette var adressert i eksempelapplikasjonene, noe som viste seg å være heller beskjedent. Jeg mente likevel at et rammeverk bør være i stand til å behandle privat kontekstanvendelse mer utfyllede.

Kapittel 8

Anvendelse av kontekstrelevant informasjon

Kontekstoppmerksomhet kan anvendes til flere ulike formål (jamfør 3.2). Jeg har valgt å fokusere på ett av de, nemlig anvendelse av kontekstrelevant informasjon. I dette kapitlet vil jeg analysere hvordan eksempelapplikasjonene gjør bruk av dette. Likt som forrige kapittel vil jeg også her skille analysen inn det i to deler: Anvendelse med utgangspunkt i applikasjonene, og anvendelse med utgangspunkt i brukerne. Begge delene skal tjene til å fange de resterende kravene til kravspesifikasjonen påbegynt i forrige kapittel. Som nevnt vil denne presenteres samlet i Kapittel 9.

8.1 Anvendelse i forhold til applikasjonene

Nedenfor vil jeg å belyse hvordan eksempelapplikasjonene gjør bruk av kontekstrelevant informasjon. Jeg vil drøfte hvordan slik informasjon blir sett på som kontekstrelevant informasjon, og behovet som da oppstår for historikk. Deretter vil jeg drøfte statiske og dynamiske aspekter ved slik informasjon, før jeg avslutter med aktivering og deaktivering.

8.1.1 Kontekstrelevant informasjon som kontekst

Formålet med kontekstrelevant informasjon er å gi brukeren relevant informasjon i henhold til situasjonen han befinner seg i. At jeg har identifisert dette, er imidlertid ikke spesielt oppsiktsvekkende; det er nettopp slik funksjonalitet jeg ønsker å rette søkelyset på i oppgaven. Likevel nevner jeg det her for ordens skyld. Mer spesielt er hvordan anvendelse av kontekstrelevant informasjon også kan være med på å beskrive brukerens situasjon: Enkelte ganger inneholder kontekstrele-

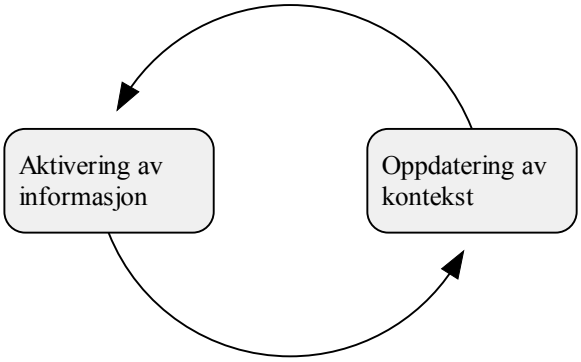
vant informasjon opplysninger som for eksempel hvor brukeren er, eller hva slags aktiviteter han bedriver ved aktivering. Dette kommer blant annet til syne i Mocado. Påfallende ofte avslører en annotering her kontekstuelle opplysninger. La oss for eksempel anta at en annotering er plassert rundt statuen Sinnataggen, og inneholder enkel informasjon av typen ”statuen ved siden av deg heter Sinnataggen”. Ved aktivering vil brukeren på normalt vis få denne informasjonen presentert. Det spesielle her er at annoteringen også sier noe om hvilke objekter (eller steder) brukeren er i nærheten av. Dette er informasjon som tilhører hans fysiske kontekst (jamfør 3.1.1), og kunne av den grunn ha blitt benyttet for å registrere hans situasjon ytterligere. Kontekstrelevant informasjon kan derfor betraktes til å inneholde to sider av samme informasjon; en for brukeren, og en for applikasjonen (se Figur 27). Jeg har valgt å benevne det første for informasjonens *brukerinformasjon*, og det siste for dens *postkontekst* (på grunn av likheten dette tilfellet har til kontekstutledning, jamfør 7.1.5).

Aktiveringskontekst	Brukerens lokasjon = "Thorvald Meyers gate"
Postkontekst	Nærmeste holdeplass = "Olaf Ryes Plass"
Brukerinformasjon	Nærmeste bussholdeplass fra der du står nå heter "Olaf Ryes Plass". For å komme dit, kan du følge veien oppover (nord) ca. 200 meter

Figur 27: Postkontekst – Figuren illustrerer et eksempel på hvordan kontekstrelevant informasjon også kan inneholde kontekstuell informasjon

Et spennende tilfelle som kan inntreffe om det ovenstående realiseres, er hvordan aktiveringsprosessen kan pågå i en sirkulær prosess. Etter at informasjon er blitt aktivert, vil dette kunne skape en oppdatering i brukerkonteksten (med bidraget fra postkonteksten). Dette kan igjen medføre at mer informasjon blir aktivert, og videre til at mer kontekst blir tilgjengelig. Slik kan applikasjonen holde på i flere sykler (se Figur 28). Dette har en spennende likhet med det jeg i forrige kapittel refererte til som kontekstutledning. Dette er når ny kontekst blir registrert fra eldre kontekst. For eksempel argumenterte jeg for at det kunne la seg gjøre å finne nærmeste bussholdeplass fra en bestemt posisjon, forutsatt at applikasjonen var supplert med kartinformasjon. Det ovenstående er omtrent det samme som kontekstutledning: Også her vil en postkontekst kunne utledes dersom en gitt prekontekst er tilfredsstilt. I dette tilfellet er informasjonens prekontekst og aktiveringskontekst samme informasjon.

Som beskrevet i Kapittel 6.3 er jeg avhengig av å kjenne til passerte sekunderingspunkter under en trening i Mobitras. Jeg har kalt dette hans *aktiveringshistorikk*. I Mocado ønsket jeg meg et lignende system for å variere informasjonen som ble presentert ved annen og tredjegangs aktivering. Cheverst hevder at applikasjoner bør gjenkjenne slike aktiveringer, fordi de beskriver viktige sider ved brukerens kontekstuelle situasjon [Cheverst et al. 2000]. Aktiveringsinformasjonen er med andre ord også kontekstuell informasjon (historisk kontekst).



Figur 28: Aktiveringsprosessen – Figuren viser hvordan en oppdatering av brukerens kontekst kan resultere i at kontekstrelevant informasjon blir aktivert. Aktiveringen, igjen, kan gi kontekstuelle opplysninger tilbake til applikasjonen, noe som kan resultere i at mer informasjon blir aktivert.

8.1.2 Behov for historikk

Over har jeg drøftet hvordan kontekstrelevant informasjon kan tilby kontekstuell informasjon. I slike tilfeller gjør også et annet behov seg gjeldene: Behovet for historikk.

> *"Her om dagen gikk jeg en tur langs Vippetangen med Mocado. Formålet var å se hvordan en slik tur ville ha fortonet seg i januar 2001. Etter å ha satt meg tilbake i tid, fikk jeg frem hvilke båter, biler, og mennesker som var i nærheten, samt hvordan disse forflyttet seg omkring. Straks ferjen mot Gressholmen la til kai (i Mocados historiske verden), ble også informasjon om dens billettpriser aktivert og vist frem."*

Om kontekstrelevant informasjon kan endres eller byttes ut over tid, kan dette resultere i at konteksthistorie går tapt. Om man i Mocado skal kunne gjenskape situasjoner bakover i tid, er man også avhengig av å anvende de annoteringene som på det aktuelle tidspunktet var tilgjengelige. Spesielt er dette viktig om de også tilbyr postkontekst som applikasjonen gjør nytte av. Da vil behovet for oppbevaring av slik informasjon bli viktig (i likhet med annen kontekstuell informasjon, jamfør 7.1.4) La oss ta et eksempel med en annotering på et anleggsområde. Denne opplyser om prosjektet som utføres på plassen, og oppdateres jevnlig etter hvert som arbeidet skrider frem. Om man ønsker å undersøke hvilken informasjon man ville ha fått om man befant seg på samme sted noen måneder i forveien, ville man også vært avhengig av å benytte riktig versjon av prosjektannoteringen.

8.1.3 Statisk og dynamisk informasjon

De fleste eksempelapplikasjonene benytter statisk kontekstrelevant informasjon. Dette er informasjon som ikke endrer seg under eksekvering. Observasjonene som forfattes i The Field assistant er et eksempel på statisk informasjon. Det samme er annoteringene i Mocado. Er en annotering først konstruert og lagret, så forblir den slik helt til noen sletter eller endrer den. I Mocado og Mobitras kom jeg likevel over et problem: I enkelte situasjoner lot det seg ikke å gjøre å forfatte statisk informasjon, den var isteden dynamisk. Det viser seg også at både aktiveringskonteksten og den assosierte informasjonen kan være nettopp det.

Dynamisk aktiveringskontekst

> *"Da jeg i sommer var ombord i båten til min far, var jeg fristet til å lage en annotering på dekk. Denne kunne gi besøkende litt informasjon om båtens navn og historie. Dessverre ville en slik annotering blitt stående igjen ved bryggekanten når vi senere kastet loss."*

Som eksempelet illustrerer har Mocado en liten begrensning når det gjelder annoteringer. Den forutsetter at disse beskriver objekter og steder som aldri endrer geografisk posisjon. Dette var også det vanligste tilfellet under uttesting; verken statuene eller parkene flyttet noen gang på seg. Problemet ville derimot ha oppstått om jeg hadde assosiert annoteringene til bevegelige objekter. Mens objektene endret lokasjon, ville annoteringene forholdt seg i ro[1]. Aktiveringskonteksten hos annoteringene tillater med andre ord ikke slike situasjoner å inntreffe.

I Mobitras kan man observere en lignende situasjon som ovenfor. Her gjelder det imidlertid skyggene. Disse oppfører seg likt som annoteringene, bortsett fra at de endrer lokasjon under en trening. Det som gjør disse spesielt interessante, er at de verken er assosiert til et sted eller et objekt. Dette er annoteringene i Mocado. For eksempel kunne annoteringen i båteksempelet ovenfor vært lokalisert relativt til båten. Dette lar seg ikke gjøre med skyggene. De beveger seg isteden uavhengig av fysiske objekter.

Dynamisk informasjon

> *"I det Ole ankommer bussholdeplassen på vei til jobben, aktiveres følgende informasjon: 'Neste buss, nummer 414 mot Oslo, er litt forsinket og ventes om cirka fire minutter."*

[1] Informasjonen er assosiert til en geografisk posisjon, og derfor uavhengig av objektet.

Det finnes også tilfeller hvor heller ikke brukerinformasjonen kan avgjøres på forhånd. Eksempelet over gir en god illustrasjon på dette. Hvor langt unna en buss er en holdeplassen kan ikke avgjøres før informasjonen aktiveres. Det samme tilfellet ser man i Mobitras: Mye av sekunderingsinformasjonen som presenteres der, er avhengig av brukerens kontekst under presentasjonen. Et eksempel på dette er kilometertiden. Denne kan naturlig nok ikke bestemmes før utøveren har tilbakelagt minst én kilometer under treningen.

I noen tilfeller kan både aktiveringskonteksten og dens assosierte informasjon være dynamisk samtidig. Dette er tilfellet hos skyggene: både deres lokasjon og hastighet vil endre seg som en funksjon av tiden. Så hvis en skyggeannotering i presenterer informasjon om sin egen hastighet, vil både aktiveringskonteksten og informasjonen endre seg kontinuerlig.

Selv om behovet for dynamiske kontekstrelevant informasjon er tilstede, er det likevel bruk av statisk informasjon som dominerer blant eksempelapplikasjonene. Jeg har allerede nevnt observasjonene i The Field assistant. Forelesningsfoilene i The Conference assistant er et annet eksempel. Disse er knyttet opp mot forelesningenes tid og sted, og presenteres for brukeren i det han ankommer lokalene. Attraksjonsinformasjonen i The Lancaster Guide blir realisert tilsvarende: Informasjonen her oppbevares som statiske HTML-sider på en webserver. Jeg vurderer derfor bruk av statisk kontekstrelevant informasjon som viktigst for et rammeverk å støtte.

8.1.4 Aktivering og deaktivering

Jeg har allerede definert begrepet aktivering (jamfør 3.3). Aktivering inntreffer når en bestemt mengde kontekstrelevant informasjon blir vurdert som gyldig og presentert for brukeren. Under utviklingen av Mocado og Mobitras oppsto også to andre tilsvarende behov: *Deaktivering* og *reaktivering*. Disse vil jeg drøfte nedenfor.

I Mocado har man ingen mulighet til å spesifisere når en annotering skal deaktiveres. Dette bestemmes isteden implisitt. Aktiveringskonteksten til en annotering består av et areal, og deaktiveringen inntreffer når man beveget seg ut av dette. Denne ”enten er man innenfor, eller så er man utenfor” løsningen viste seg likevel som litt for enkel, spesielt når det angikk grenseproblematikken (jamfør 5.4 og 6.3). Dette problemet inntreffer hvis brukeren beveger seg langsetter grensen til en annotering, slik at annoteringen blir aktivert og deaktivert hyppig om hverandre.

I Mobitras oppsto grenseproblematikken med skyggene. Her avslørte problemet seg enda raskere. Siden det er relativ vanlig å ligge noenlunde likt med en skygge under et løp, blir man også relativt ofte løpende inn og ut av deres annotering. Spesielt gjelder dette om skyggene er basert på opptak gjort få dager i forveien (prestasjonene varierer gjerne ikke så mye over bare noen få dager). Løsningen jeg kom frem til her, var å introdusere et eget felt benevnt *deaktiveringskontekst*.

Deaktiveringskonteksten beskriver situasjonen som må oppstå for at en bestemt mengde kontekstrelevant informasjon ikke lenger skulle oppfattes som gyldig. I Mobitras har jeg satt radiusen i deaktiveringskonteksten hos skyggeannoteringene til å være fem meter større en tilfellet er i aktiveringskonteksten. Slik tvinger jeg utøveren litt unna skyggen før dens annotering kan aktiveres på nytt. Grenseproblematikken er for øvrig ikke omtalt hos noen av de andre eksempelapplikasjonene jeg studerte. Likevel vil jeg hevde at dette problemet også vil gjelde de fleste av disse fordi de anvender kontekstrelevant informasjon relativt likt som Mocado og Mobitras.

Ovenfor introduserte jeg begrepet *reaktivering*. Reaktivering vil jeg definere til å være aktivering av kontekstrelevant informasjon som av samme brukeren er blitt aktivert og deaktivert en eller flere ganger tidligere. I Mocado er det et behov for å kunne variere informasjonen som blir presentert ved reaktiveringer. Det vil si, informasjonen ved første aktivering av en annotering ønskes ofte mer detaljert enn ved aktivering nummer to. Dette lar seg dessverre ikke gjøre i Mocado; den er rett og slett ikke konstruert for å være oppmerksom på brukerens aktiveringshistorikk.

Ovenfor drøftet jeg hvordan en separat deaktiveringskontekst er viktig for å løse grenseproblematikken. Imidlertid eksisterer det også andre fordeler ved en slik spesifikasjon, noe som blant annet kommer dette til syne i Mocado. Her har jeg enkelte ganger hatt ønske om å bevare en annotering aktiv selv lenge etter at jeg har forlatt dens lokasjon. Dette er for eksempel relevant om jeg spaserer mens jeg leser annoteringens informasjon. I slike tilfeller er det uheldig om den plutselig blir deaktivert. Dette ønsket blir også enda mer relevant om informasjonen tilbyr postkontekst til applikasjonen (jamfør 8.1.1). Postkonteksten kan for eksempel fortelle om statuen brukeren observerer, informasjon som tilhører hans fysiske kontekst (jamfør 3.1.1). Likevel kan postkonteksten også fortelle mer; bare det at brukeren har besøkt statuen en eller annen gang, er også med på å beskrive hans historiske kontekst. Ved å la applikasjonen ivareta og anvende denne informasjonen aktivt, har den også mulighet til å avgjøre om en senere aktivering egentlig er en reaktivering.

I eksempelet ovenfor tok jeg det for gitt at Mocado oppbevarte postkonteksten fra statuens annotering, selv lenge etter at den var deaktivert. Dette måtte den jo, skulle den ha noen mulighet til å anvende disse opplysningene under en reaktivering. Sagt på en annen måte så kunne postkonteksten umulig ha den samme deaktiveringskonteksten som brukerinformasjonen. Brukerinformasjonen ble isteden deaktivert straks brukeren kom unna statuens område. Slik jeg vurderer situasjonen, burde en annotering derfor ha muligheten til å operere med flere ulike deaktiveringskontekster, i hvert fall minst to: En tilpasset postkonteksten, og en tilpasset brukerinformasjonen. Videre burde postkonteksten også kunne deles opp på en slik måte at man kan spesifisere flere ulike deaktiveringskontekster bare for den. I eksempelet ovenfor kan for eksempel postkonteksten deles inn to bidrag: Et som opplyser om statuens nærvær (fysisk kontekst), og ett som opplyser om aktiveringen (til bruk for aktiveringshistorikk). Det første kan ha en deaktiveringskontekst på lik linje med brukerinformasjonen, mens det siste kan

ha en egen deaktiveringskontekst som lar informasjonen være gyldig i ett år fremover. Om brukeren besøker den samme statuen innen dette året er omme, vil aktiveringen isteden bli tolket som en reaktivering.

8.2 Anvendelse i forhold til brukerne

Den andre delen av dette kapitlet tar utgangspunkt i brukerne, og da anvendelse av kontekstrelevant informasjon med de som utgangspunkt. Jeg vil begynne med å drøfte aspekter rundt forfatting og distribusjon, og undersøke hvilke krav som stilles til dette. Jeg vil også belyse private og offentlige aspekter i forhold til distribusjon av slik informasjon, og videre behovet for uavhengig informasjonstilgjengelighet. Jeg vil begynne med å ta for meg forholdene rundt forfatting og distribusjon.

8.2.1 Forfatting og distribusjon

Ett av de første kravene som ble fremsatt da jeg gikk i gang med Mocado, var behovet for enkel forfatting og rask distribusjon. Brukerne skulle på en relativt grei måte kunne forfatte kontekstrelevant informasjon der de var, når de var der. Pascoe hevder at enkel observasjonsinnlegging også er avgjørende i The Field Guide for at ikke applikasjonen skal føles som en hindring under arbeidet [Pascoe 1998]. For å realisere dette ble observasjonene her ferdigutfylt med kontekstuell informasjon som blant annet deres posisjon. Denne teknikken benyttes også i Mobitras: Her blir mye av aktiveringskonteksten og brukerinformasjonen til et sekunderingspunkt ferdigutfylt under løypekonstruksjonen. Jeg antar i applikasjonen at punktene vil inneholde omtrent den samme informasjonen fra punkt til punkt, slik at hvis en utøver har bestemt at hans distanse skal leses opp ved et gitt sekunderingspunkt, så vil dette sannsynligvis også være ønskelig for de påfølgende punktene.

Brown belyser kravet om enkel forfatting og distribusjon. Han hevder at dette var selve hovedmotivasjonen bak arbeidet med Stick-e notes[1] [Brown 1995]. Enkel forfatting tilsier at personer med liten eller ingen kunnskap om programmering skal kunne delta i forfattingen [Brown 1995]. Ofte er det forskjellige mennesker som utvikler applikasjonene i forhold til sluttbrukerne, og et slikt utgangspunkt er av den grunn fornuftig. Et viktig poeng er å derfor å skille den kontekstrelevante informasjonen mest mulig fra applikasjonene. Brown har videre identifisert seks krav han mener bør ivaretas angående anvendelse av kontekstrelevant informasjon [Brown 1998b]:

[1] Stick-e notes er en teknologi for anvendelse av kontekstrelevant informasjon (jamfør 10.2). Field Guide er blant applikasjonene som er utviklet med forankring i denne.

1. En bruker skal ikke behøve å ha noen forståelse av hvordan kontekstrelevant informasjon aktiveres i applikasjonen for å kunne anvende det.
2. Både brukere og profesjonelle forfattere bør være i stand til å publisere og utveksle kontekstrelevant informasjon.
3. Om en ny sensor legges til i systemet, skal applikasjonen fortsette å aktivere informasjon som ikke har noe med denne å gjøre, likt som før. Informasjon som derimot benytter den nye sensoren, skal filtreres opp mot dennes observasjoner under aktiveringsprosessen. Dette vil si at sensoren kan komme med kontekstuelle opplysninger som hindrer aktivering av informasjon som ellers ville ha blitt aktivert.
4. Nye sensorer i systemet bør ikke tvinge forfattere til å endre allerede eksisterende informasjon.
5. Kontekstrelevant informasjon bør helst ikke assosieres til kontekst som bare kan registreres av én bestemt sensor.
6. Selv om noen av sensorene i systemet skulle slutte å virke, bør systemet kunne fortsette aktiveringsprosessen på en fornuftig måte[1].

Kontekstrelevant informasjon kan være så mye: Tekst, bilde, lyd, film, eller ulike blandinger og sammensettinger av dette. I Mocado anvender jeg for eksempel tekst, bilde, og lyd i en annotering, mens jeg i sekunderingspunktene i Mobitras kun benytter lyd. Slik informasjon hevder Lieberman spesifiseres av brukeren eksplisitt, i motsetning til kontekst, som isteden registreres implisitt [Lieberman et al. 2000]. Hvordan eksplisitt informasjon kan forfattes, vil jeg imidlertid ikke belyse i denne oppgaven. Derfor vurderer jeg heller ikke enkel forfatting som et relevant krav for rammeverket jeg skal frem til.

Når det gjelder det andre kravet, rask distribusjon, så vurderer jeg dette annerledes enn enkel forfatting. Rask distribusjon kan, og bør realiseres av et rammeverk. Cheverst, for eksempel, vurderer rask distribusjon i The Lancaster Guide som viktig. Slik informasjon, hevder han, burde umiddelbart bli tilgjengeliggjort i applikasjonen dersom irritasjon hos brukerne skal forhindres.

8.2.2 Privat og offentlig tilgjenglighet

I Kapittel 7.2.3 drøftet jeg forholdet mellom privat og offentlig kontekst. Der kom jeg frem til at en bruker bør kunne begrense andres mulighet til å anvende hans egen kontekstuelle informasjon. Spørsmålet jeg vil undersøke her, er om dette også gjelder for kontekstrelevant informasjon. Brown besvarer dette, og hevder at det bør være mulig å forfatte informasjon som ikke har tilgjengelighet

[1] Hva som derimot vurderes som ”fornuftig” blir av Brown ikke beskrevet.

for andre en brukeren selv [Brown 1995][1]. Slik informasjon anvendes gjerne som et hjelpemiddel for å huske steder og ting i brukerens egen hverdag, og blir av den grunn ofte personlige [Brown 1995].

På tross av ønsket om privat oppbevaring av informasjon, er behovet for informasjonsutveksling også til stede. Dette var noe jeg planla å implementere i Mocado; muligheten for å la flere brukere se hverandres annoteringer. I Mobitras er dette realisert: Både løypene og treningsopptakene som blir produsert er tilrettelagt for enkel utveksling (uten bruk av nettverk). Likevel har også hver enkelt utøver mulighet til å begrense denne utvekslingen om ønskelig. I The Lancaster Guide blir derimot dette foretatt automatisk. Ved hjelp av et trådløst nettverk blir turistenes forfattede dokumenter gjort tilgjengelig for alle.

Hos mange av eksempelapplikasjonene er ikke PDA-brukerne de største informasjonsforfatterne. Ofte er mye av informasjonen lagt inn i systemet på forhånd. Dette er blant annet tilfellet i The Lancaster Guide. Her er de fleste severdighetene i byen blitt utplassert på egne servere i nettverket, tilgjengelig for alle. Det samme er tilfellet i The Conference assistant. Også her blir informasjon omkring forelesninger og aktiviteter lagt inn på forhånd. I tillegg blir mye av informasjon installert på brukernes maskiner når de ankommer konferansen. Det er med andre ord også et behov for distribusjon av informasjon uavhengig av informasjonen fra brukerne.

Det viser seg at det er et behov for å atskille kontekstrelevant informasjon etter innhold. For eksempel kan informasjon som beskriver turistattraksjoner samles under kategorien turistinformasjon. Annoteringer som omhandler kloakkanlegg, kan tilhøre en annen. Likt som for spørsmålet om privat og offentlig informasjon, er det også her behov for tilgangsbegrensninger. Som turist ønsker man vanligvis kun å få frem informasjon om attraksjoner, ikke det underliggende kloakkanlegget.

Også i Mobitras er det et lignende behov som det beskrevet ovenfor. Her gjelder det imidlertid muligheten for å kunne skille mellom sekunderingspunkter fra overlappende løyper. Selv om to løyper ligger omtrent på samme sted, skal de selvsagt behandles atskilt under trening. Brown argumenterer for at brukeren selv bør kunne velge hvilke kategorier han ønsker å benytte i en turistapplikasjon [Brown 1995]. Dette blir analogt med at utøveren selv kan velge hvilken løype han i Mobitras ønsker å løpe. I slike tilfeller er imidlertid ikke motivasjonen å begrense informasjonstilgjengeligheten av privatmessige grunner. Man ønsker snarere å strukturere informasjonen etter innhold. Hva slags innhold en bruker har behov for, er igjen relatert til hva slags kontekst han befinner seg i. Slik sett kan kontekstrelevant informasjon også separeres ved å la slik informasjon ha kategorien de tilhører spesifisert i deres aktiveringskontekst. For eksempel kan et sekunderingspunkt i Mobitras selv inneholde informasjon om hvilken løype den tilhører. Om utøveren har ”Sognsvann rundt” oppsatt som løype[2], så skal bare

[1] Brown uttaler seg her i sammenheng med Stick-e note systemet (se også 10.2). Dette en teknologi for å behandle kontekstrelevant informasjon.

[2] Dette er i så fall en del av hans aktivitetskontekst (jamfør 3.1.1).

sekunderingspunkter som har denne løypen spesifisert i aktiveringskonteksten, benyttes.

8.2.3 Uavhengig tilgjengelighet

Formålet med kontekstrelevant informasjon er å gi brukeren relevant informasjon i henhold til hans situasjon. Når man i The Lancaster Guide befinner seg foran Lancaster slott, skal informasjon om slottet presentert. Når man ankommer en forelesning i The Conference assistant, skal man få fremlagt forelesningsfoilene for foredraget. Klemke hevder at det er to grunner til at man assosierer kontekst til informasjon. For det første har man mulighet for å filtrere vekk informasjon som ikke er interessant for brukeren, og for det andre blir informasjonen beriket med kontekstuelle opplysninger [Klemke et al, 2001]. I denne oppgaven benytter jeg kontekstassosieringen kun til det første. Likevel ser jeg også et behov for uavhengig informasjonstilgjengelighet. Det vil si, tilgang til å anvende informasjon uavhengig av kontekst. Cheverst, med flere, at det er viktig å ha tilgang til informasjon plassert på andre steder enn der man selv er [Cheverst et al. 2000, Brown 1998b]. Han forteller at de i The Lancaster Guide ikke tillot dette i begynnelsen. Resultatet var frustrasjon, spesielt siden det ikke lot seg gjøre å få frem opplysninger om attraksjoner de så lengre unna (som for eksempel Lancaster slott, som var synlig fra relativt lange avstander).

I The Field assistant er fri tilgang til kontekstrelevant informasjon implementert. Her kan man, om ønskelig, legge til observasjoner helt uavhengig av sin egen situasjon. På samme vis har deltagerne i The Conference assistant også mulighet til å hente frem opplysninger for enhver forelesning på konferansen. Også i Mocado gjorde jeg de samme erfaringene. Her fant jeg det særdeles nyttig å kunne forfatte annoteringer for steder man så lengre unna. I noen tilfeller var dette også helt nødvendig. Om man for eksempel skulle lage en annotering for slottet, måtte man i prinsippet ha stilt seg midt i bygget og spesifisert en radius som dekket hele plassen. Siden ikke hvem som helst har adgang dit, er det langt enklere å kunne markere av et slikt område på kartet. Oppsummert vil jeg derfor hevde at man bør ha tilgang til å både forfatte og motta informasjon for andre situasjoner enn sin egen.

8.3 Oppsummering

I dette kapitlet har jeg drøftet behandling av kontekstrelevant informasjon. Jeg har sett på to forhold; behandling i forhold til applikasjonene, og behandling i forhold til brukerne. Jeg begynte således med applikasjonene. Det viste seg her at kontekstrelevant informasjon kunne bidra med mer enn bare brukerinformasjon. Ofte inneholdt informasjonen mange kontekstuelle opplysninger. Jeg drøftet i tillegg hvordan dette kunne utnyttes for å registrere brukerens kontekstuelle situasjon ytterligere.

Når kontekstrelevant informasjon var en kilde til kontekstuell informasjon, oppsto et behov for oppbevaring av slik informasjon i en historikk. Dette var nødvendig for å tillate situasjonssimulering. I tillegg drøftet jeg anvendelse av aktiveringshistorikk. Dette var hendig om man ønsket presentasjon av variert informasjon ved reaktiveringer.

Jeg identifiserte også et behov for anvendelse av dynamisk informasjon. Dette var informasjon hvor enten aktiveringskonteksten, postkonteksten, eller brukerinformasjonen endret seg under eksekvering. Videre definerte jeg statisk informasjon til å være et spesialtilfelle av dette, der alle elementene ovenfor forhold seg i ro. Det siste ble mest anvendt blant eksempelapplikasjonene.

Det viste seg at man også hadde et behov for å la kontekstrelevant informasjon inneholde flere ulike deaktiveringskontekster. Postkonteksten kunne blant annet med fordel operere med flere ulike variasjoner over denne.

Etter å ha drøftet behandling av kontekstrelevant informasjon med fokus på applikasjonene, belyste jeg det samme temaet med søkelyset rettet mot brukerne. Jeg begynte med enkel forfatting og rask distribusjon. Det første vurderte jeg til å være en del av brukerens eksplisitte interaksjonen med en applikasjon, og derfor ikke et krav for et rammeverk. Imidlertid betraktet jeg enkel distribusjon som meget relevant.

Etter forfatting og distribusjon drøftet jeg forholdet mellom privat og offentlig informasjonstilgjengelighet. Kontekstrelevant informasjon kunne inneholde en del personlige opplysninger, noe som indikerer et behov for privat oppbevaring. Likevel ble ønsket om informasjonsutveksling også belyst. I enkelte av applikasjonene var i tillegg mye av informasjonen forfattet av andre enn brukerne, og gjort felles for de alle ved hjelp av egne servere i nettverket. Et eksempel på dette var HTML-dokumentene hos The Lancaster Guide.

Til sist så jeg på behovet for uavhengig tilgjengelighet. Selv om kontekstoppmerksomhet ble benyttet for å filtrere vekk urelevant informasjon, var det også et ønske om å kunne forfatte og lese slik informasjon uavhengig av situasjon. Jeg mener derfor at brukernes tilgang til kontekstrelevant informasjon bør være uavhengig av deres kontekst.

Kapittel 9

Kravspesifikasjon

I de to foregående kapitlene har jeg identifisert generell og relevant funksjonalitet blant applikasjonene rammeverket skal støtte. Dette innebar først fremst hva som var blitt implementert, men også utvidelser og mangler. Til sammen har dette ledet meg frem til en samlet kravspesifikasjon for et rammeverk. Denne er hva jeg også vil presentere her.

Strengt tatt kommer jeg ikke med noe nytt i dette kapitlet. Det er heller ikke meningen. Alle kravene jeg fremsetter er allerede presentert og drøftet tidligere, i de to kapitlene føre. Dette blir mer en slags konkretisering. Imidlertid har jeg strukturert kravene på en litt mer oversiktlig måte. Dette gjør det enklere å være mer presis. Presis når jeg i neste Kapittel drøfter andre alternative løsningsforslag på rammeverk, og presis når jeg til sist drøfter mitt eget rammeverk, Condor.

Listen under teller til sammen tjue krav. Likevel vektlegger jeg ikke alle likt. Jeg vurderer for eksempel anvendelse av eksterne sensorer, krav 1, som langt viktigere enn kontekstoverstyring, krav 8. Derfor kan jeg også vurdere et rammeverksforslag i neste Kapittel som meget bra, selv om bare et fåtall av kravene blir tilfredsstilt.

Kravene kan konkretiseres og struktureres på følgende måte:

9.1 Anvendelse av kontekstoppmerksomhet

Krav 1: Anvendelse av eksterne sensorsystemer – Rammeverket bør ha et generelt system for anvendelse og gjenbruk av eksterne sensorsystemer. Dette er systemer som registrerer kontekst ved hjelp av eksternt tilkoblet utstyr. Denne metoden kan utvinne en del nøkkelkontekst.

Krav 2: Anvendelse av interne sensorsystemer – Rammeverket bør ha et generelt system for anvendelse av interne sensorsystemer. Dette er systemer som registrerer kontekst *uten* hjelp av eksternt tilkoblet utstyr. Også denne metoden kan utvinne en del nøkkelkontekst.

Krav 3: Betinget anvendelse av sensorsystemer – Også en sensorløsning har en kontekstspesifikasjon. Denne beskriver under hvilke forhold sensoren kan levere gyldige observasjoner. Bare sensorsystemer som dekker brukerens kontekst under eksekvering bør anvendes.

Krav 4: Dynamisk skifte av sensorsystemer – Et rammeverk bør støtte dynamisk skifte av sensorsystemer. Dette er muligheten for å anvende observasjoner fra de til enhver tid beste sensorene. Slike skifter burde helst foregå automatisk innen rammeverket, og da etter følgende tjenestekvalitetskrav: dekning, nøyaktighet, stabilitet, oppløsning, oppdateringsfrekvens, og reaksjonstid.

Krav 5: Kommunikasjon av kontekst – Et rammeverk bør kunne behandle flere ulike brukere fra flere ulike klientapplikasjoner, samtidig. Mange av disse applikasjonene kan være konstruert for å knytte seg sterkt til en bruker, og kommunikasjon mellom klientapplikasjonene kan derfor utvinne sosial kontekst for hver enkelt.

Krav 6: Privat kontekstanvendelse – Om kommunikasjon av kontekst er mulig, bør man også ha mulighet for å helt, eller delvis, begrense denne distribusjonen under eksekvering. Mye av en brukers kontekst kan til ulike tider, og ovenfor ulike personer, være av sensitiv art, slik at dette ikke ønskes utvekslet fritt.

Krav 7: Manuell kontekstspesifikasjon – Et rammeverk bør tillate manuell kontekstspesifikasjon. En del kontekstuell informasjon er vanskelig å registrere automatisk, og brukerne (gjennom klientapplikasjonene) må i enkelte tilfeller spesifisere dette selv.

Krav 8: Kontekstoverstyring – En klientapplikasjon bør ha mulighet for å overstyre hele, eller deler av en brukers kontekst. Ofte kan brukerne (gjennom klientapplikasjonene) ønske å sette seg i andre situasjoner enn den systemet mener de befinner seg i.

Krav 9: Anvendelse av konteksthistorikk som kontekst – Et rammeverk bør oppbevare tidligere registrert kontekst, og anvende dette under aktiveringsprosessen. Hvilke situasjoner brukeren har opplevd i fortid kan være med på å utdype hans situasjon i nåtid.

Krav 10: Kontekstutledning – Ved hjelp av kontekstutledning kan en del kontekst på et høyere, ikke-sansbart, nivå kan utledes. Et rammeverk bør av den grunn være i stand til å gjennomføre en slik utledning automatisk. Dette inkluderer også kontekstomforming og repeterende kontekstutledning.

Krav 11: Abonnering og spørring etter kontekst – En klientapplikasjon bør ha fri tilgang til sin egen brukers kontekst. Selv om kontekstuelle opplysninger hovedsakelig er tiltenkt aktiveringsprosessen, kan den i seg selv være informativ også for applikasjonen (og brukeren).

9.2 Anvendelse av kontekstrelevant informasjon

Krav 12: Anvendelse av kontekstrelevant informasjon – Et rammeverk skal ha et system for behandle kontekstrelevant informasjon. Slik informasjon skal kunne aktiveres ut ifra brukerens kontekst.

Krav 13: Anvendelse av kontekst fra kontekstrelevant informasjon – Mye kontekstrelevant informasjon som i utgangspunktet er tiltenkt brukeren, kan også bidra med å beskrive hans kontekstuelle situasjon. Slik informasjon bør derfor anvendes under aktiveringsprosessen. Et rammeverk bør derfor ha et system for å hente ut kontekstuelle opplysninger hos aktivert informasjon.

Krav 14: Dynamisk informasjon – Et rammeverk bør tillate dynamisk informasjon. Både aktiveringskonteksten, deaktiveringskontekstene, kontekstbidragene, og brukerinformasjonen kan man i enkelte tilfeller endres under eksekvering.

Krav 15: Anvendelse av historikk – Hva slags informasjon som av en bruker er aktivert tidligere, kan betraktes som en del av hans historiske kontekst. Et rammeverk bør av den grunn ha et system for å ivareta, og anvende, aktiveringshistorikk. Om kontekstuelle opplysninger blir uthentet fra informasjonen (postkontekst), eller informasjon i seg selv kan være dynamisk, bør også informasjonen ivaretas tilsvarende.

Krav 16: Separate deaktiveringskontekster – Aktivert informasjon kan ønskes deaktivert uavhengig av dens aktiveringskontekst. Av den grunn bør man kunne la en bestemt mengde informasjon kunne operere med en eller flere separate deaktiveringskontekster.

Krav 17: Informasjonsforfatting – Et rammeverk bør tillatte både lagring og endring av kontekstrelevant informasjon under eksekvering. Slike endringer bør gjøres gjeldene i rammeverket umiddelbart etter lagringen/endringen.

Krav 18: Distribusjon – Et rammeverk bør tillate utveksling av kontekstrelevant informasjon mellom flere brukere. Slik informasjon har ofte relevans for flere enn forfatteren selv.

Krav 19: Privat tilgjengelighet – En klientapplikasjon bør ha mulighet for å helt eller delvis begrense distribusjonen av kontekstrelevant informasjon under eksekvering. Slik informasjon kan inneholde opplysninger av sensitiv art, og derfor ikke ønsket utvekslet fritt.

Krav 20: Uavhengig tilgjengelighet – En klientapplikasjon bør ha tilgang til å anvende kontekstrelevant informasjon uavhengig av kontekst. Kontekstoppmerksomhet er snarere et verktøy for å filtrere vekk irrelevant informasjon, fremfor en måte å begrense slik informasjons tilgjengelighet.

Kapittel 10

Tre eksterne rammeverk

I forrige Kapittel avsluttet jeg analysen av rammeverksområdet. Dette var den første fasen ved en *a priori* tilnærming til et rammeverk (jamfør 2.2). Målet var å studere oppgavens område relativt detaljert, for på den måten å komme frem til en kravspesifikasjon for et rammeverk. Denne vil jeg benytte meg av her.

Formålet med dette kapitlet er å vurdere hva andre har gjort før meg. Jeg vil undersøke om det eksisterer noen rammeverk i dag som kunne vært anvendt for å utvikle oppgavens applikasjoner. For å svare på dette vil jeg vurdere tre av de mest relevante løsningene jeg har funnet opp mot kravspesifikasjonen. Kriteriene for valg av løsninger har vært tilgjengelighet til arkitekturbeskrivelse, og hvorvidt de faktisk har blitt implementert og utprøvd (ikke bare skissert).

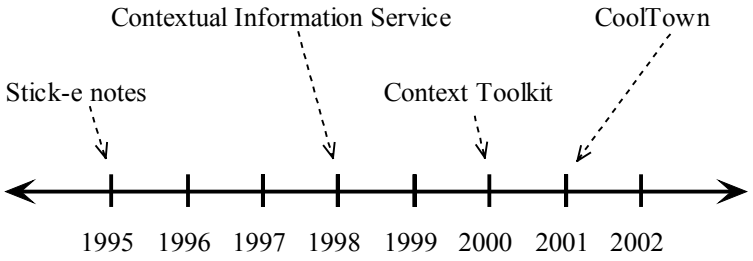
Selv om rammeverksløsningene kan vise seg å støtte kravspesifikasjonen delvis, kan de likevel adressere noen av kravene hver for seg. I slike tilfeller er det også interessant å studere deres konstruksjon, og hvordan disse kravene blitt realisert. Noe av hensikten med kapitlet er derfor å vurdere også dette. Implementasjonsløsninger jeg finner relevante og fornuftige, vil jeg ta med meg og benytte som utgangspunkt og inspirasjon for Condor.

Kravspesifikasjonen inneholder nokså mange krav. Av den grunn vil jeg bare gjøre rede for dem som i mer eller mindre grad er blitt innfridd. Siden ett av formålene med analysen er gjenbruk av design, vurderer jeg innfridde krav som mer interessant en det motsatte; hvilke krav som ikke er blitt besvart. Jeg vektlegger med andre ord designgjenbruk som et viktigere spørsmål enn hvorvidt rammeverkene støtter oppgavens applikasjoner. De kravene jeg unnlater å gjøre rede for har jeg med andre ord vurdert til ikke å bli tilfredsstilt.

Noen av kravene i kravspesifikasjonen er også av en slik natur at de ikke nødvendigvis krever egne komponenter for å la seg løse. Et eksempel på dette er anvendelse av interne sensorer. Hos mange programmeringsspråk (med et godt utbygd API) kan en del slikt registreres ved hjelp av få systemkall. Dette er for eksempel tilfellet med kontekstkategorien tid og dato i JDK 1.3. Hvilke krav jeg velger å

drøfte avhenger derfor av hva som kan leses ut fra arkitekturene direkte, samt opplysninger som har kommet ved hjelp av kildene jeg har anvendt.

To av rammeverkene jeg vil studere er i mine kilder beskrevet relativt overordnet. Av den grunn kan jeg også unngå å få med meg detaljer som ellers ville rettferdiggjort avskrevne krav. Dette kan videre medføre at resultatene jeg kommer frem til også blir feilaktige. Jeg vil kommentere dette der jeg vurderer det nødvendig. Løsningene jeg har valgt å studere er: Stick-e notes og Contextual Information Service (vurderes sammen), Context Toolkit, og CoolTown (se Figur 29).



Figur 29: Tidslinje over eksterne rammeverk – Tidslinjen viser når rammeverkene jeg har studert ble utviklet.

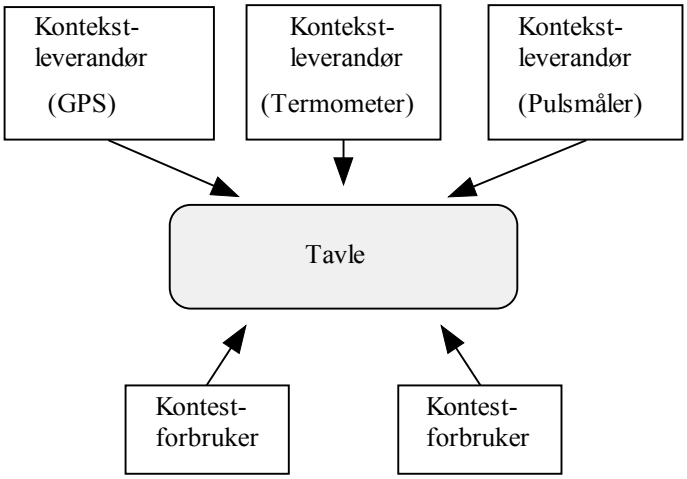
Før jeg begynner drøftingen, ønsker jeg imidlertid å gjøre kort rede for to ulike kategorier av rammeverk innenfor oppgavens område. Disse divergerer i forhold til hvordan verden beskues.

10.1 Hvordan verden kan beskues

Rammeverk for kontekstoppmerksomme applikasjoner kan gjerne kategoriseres etter hvordan de modellerer verden. Mens noen løsninger konsentrerer seg om kun én bruker, tar andre høyde for at flere slike kan eksistere parallelt. Dette gir også opphav til to ulike konstruksjonsmodeller; henholdsvis *tavleløsningen*, og *"virkelig verden"* [Dey 2000]. Grovt sett kan alle de tre løsningene jeg vil studere i dette kapitlet kategoriseres under en av disse.

Tavleløsningen bygger på en idé om at det skal eksistere en slags digital tavle innad i rammeverket (se Figur 30). Denne skal oppbevare brukerens kontekstuelle tilstand. Alle komponenter som ønsker å spørre eller abonnere etter kontekstuell informasjon (*kontekstforbrukere*), gjør dette ved å aksessere tavlen. På den måten trenger de ikke ha kjennskap til *kontekstleverandørene* i systemet, kun tavlen. Begrepet "kontekstleverandør" er en fellesbetegnelse jeg vil benytte for alle komponenter som publiserer kontekstuell informasjon i et rammeverk. Et sensorsystem er et eksempel på en slik leverandør. I tavleløsningen leverer kontekstleverandørene sine kontekstuelle opplysninger direkte til tavlen, fremfor til kontekstforbrukerne. Dermed trenger de ikke ha noe forhold til hvem som skal anvende informasjonen. Tavlen virker med andre ord som en mediator mellom kon-

tekstforbrukerne og kontekstleverandørene. Til sammen fjerner dette den ellers så tette koblingen mellom sensorer og applikasjoner som ofte er til stede hos kontekstoppmerksomme applikasjoner [Dey 2000, Pascoe 1998].



Figur 30: Tavleløsningen – Tavlen fungerer som en mediator mellom de som tilbyr kontekst, og de som anvender den.

Selv om tavleløsningen kan adressere mange av kravene i kravspesifikasjonen, har den likevel en liten ulempe. Den har vanskelig for å skille klart mellom flere brukere [Pascoe 1998, Dey 2000]. Om en applikasjon for eksempel ønsker å observere to personer samtidig, trenger den å kunne skille mellom hvilken kontekst som tilhører den ene, og hvilken som isteden tilhører den andre. Dette kan fort bli vanskelig og tungvint om kontekstuell informasjon fra begge skal ende opp sammenblandet på samme sted. Dette motiverer isteden for den andre metoden, ”virkelig verden”.

”Virkelig verden” har som utgangspunkt å modellere systemet etter hvordan verden egentlig fungerer. Med dette menes at flere brukere kan eksistere og samtidig ha hver sin individuelle kontekst [Dey 2000, Pascoe 1998]. Pascoe hevder for eksempel at lokasjonsdata i seg selv ikke gir noen mening med mindre dataene assosieres til et objekt [Pascoe 1998]. I tavleløsningen ligger gjerne denne assosiasjonen litt implisitt ved at tavlen tilhører én bruker, og bare han. I ”virkelig verden” må systemet isteden være i stand til å behandle flere brukere samtidig, og videre kunne skille mellom hvilken kontekst som tilhører hver enkelt bruker [Dey 2000]. Til dette blir tavleløsningen litt for enkel.

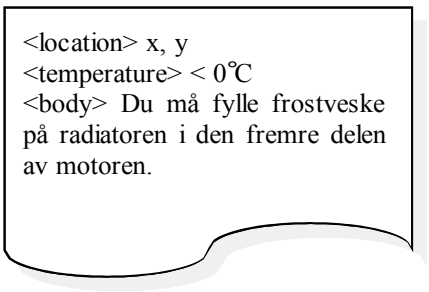
Selv om tavleløsningen representerer en litt enklere løsning enn hva ”virkelig verden” gjør [Dey 2000], er den ikke nødvendigvis dårligere. Dette kommer av at det i mange tilfeller kun er én person som er interessant, nemlig brukeren. Dette var tilfellet i Mobitras og Field assistant. Blant slike applikasjoner, der brukeren er eneste entitet av interesse, kan det være tilstrekkelig å implementere tavleløsningen. Jeg vil imidlertid hevde, som beskrevet i krav 5, at et generelt rammeverk bør være i stand til takle flere personer samtidig. I både Conference Assistant, Lancaster Guide, og Mocado var dette implementert eller ønsket.

10.2 Stick-e notes og Contextual Information Service

Den første rammeverket jeg vil studere kommer fra universitetet i Kent i Canterbury. Her har de utviklet Stick-e notes og Contextual Information Service (CIS) [Brown 1995, Brown 1998b, Pascoe 1998]. Mens Stick-e notes er en teknologi for å behandle kontekstrelevant informasjon[1], er CIS en arkitektur for å behandle kontekstoppmerksomhet.

Stick-e notes er strengt tatt ikke et rammeverk. Isteden er det en teknologi for å beskrive kontekstrelevant informasjon. Brown ønsker hovedsaklig å gjøre forfatting av slik informasjon enkel, og har selv hentet inspirasjon fra hvordan HTML-dokumenter forfattes og publiseres på Internett i dag [Brown 1995]. Her kan stadig flere lage sine egne hjemmesider, selv uten dypere kunnskaper om verken Internett, nettlesere, eller programmering.

En Stick-e note har mange likheter med de mer kjente og fysiske *post-it lappene*. Disse kan inneholde tekstlig informasjon, og bli festet til en eller annen lokasjon ved hjelp av litt lim. En Stick-e note kan imidlertid festes til mer enn bare lokasjon. I prinsippet kan de assosieres til en hvilken som helst situasjon. En Stick-e note består av to felter; et som forteller når notatet skal aktiveres (en aktiveringskontekst), og et med dens tilhørende brukerinformasjon. Formatet på dokumentet er spesifisert i SGML, et format som kan tenkes på som en forhenværende versjon av XML (se Figur 31).

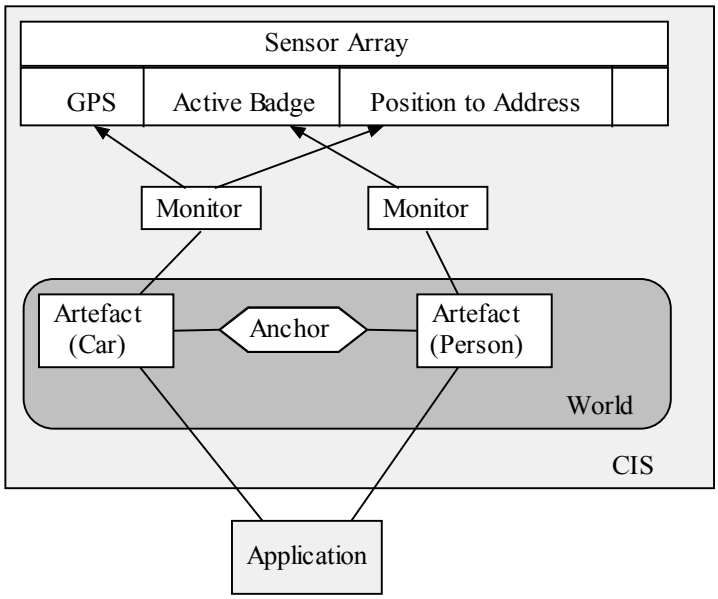


Figur 31: En Stick-e note – Når brukeren befinner seg på posisjon x (som er ved en motor) og temperaturen i luften er mindre enn 0ºC, så skal brukeren få beskjed om å fylle frostveske på motoren.

Som et supplement til Stick-e notes har Pascoe utviklet CIS [Pascoe 1998]. CIS er et forslag på en arkitektur for å behandle kontekstoppmerksomhet. CIS består hovedsakelig av fem elementer: *artifacts*, *world*, *sensor array*, *monitors*, og *anchors*. Blant disse er artefaktene (*artifacts*) mest sentrale. En artefakt kan representere en person, en gjenstand, et dyr, et sted, eller lignende. Disse har hver for seg et navn, en typebestemmelse (person, bil, dyr, eller lignende), og en kontekst-tilstand. Det siste er en beskrivelse av artefaktens kontekstuelle situasjon, og kan for eksempel inkludere dens lokasjon. Alle artefaktene oppbevares i en struktur

[1] Field Assistant (jamfør 4.2) er en av applikasjonene utviklet ved hjelp av Stick-e notes.

kalt *world*. Når en klientapplikasjonene henter frem informasjon om en artefakt, eller legger til noen egne artefakter selv, gjøres dette via denne (se Figur 32).



Figur 32: CIS – En overordnet arkitektur. GPS og Active Badge er sensorkomponenter, mens "GPS to Address" er en synthesizer.

For å knytte artefaktene opp mot den fysiske verdenen, anvendes en struktur kalt *sensor array*. Denne oppbevarer komponenter av to slag: *sensors* og *synthesizers*. De første tar seg av å kommunisere med eksterne sensorer tilkoblet systemet, mens de andre utleder ny kontekst fra artefaktenes tilstand. En persons lokasjon kombinert med tid kan for eksempel anvendes av en *synthesizer* for å bestemme høyden på tidevannet der han befinner seg.

Sensorkomponentene knyttes opp mot artefakter i systemet via *monitors*. Disse tar seg av å oppdatere artefaktenes kontekstuelle tilstand, og kan bytte mellom å anvende forskjellige sensorløsninger i henhold til tjenestekvalitet. I tillegg er monitorene konstruert for å ivareta en konteksthistorie over mottatte data. Denne historien anvendes for å simulere forventede observasjoner om noen av sensorene skulle stoppe å virke for kortere perioder.

Til sist benytter CIS *anchors*. Dette er komponenter som dynamisk binder relaterte artefakter sammen. En artefakt kan for eksempel assosieres til et anker som knytter seg til andre artefaktkomponenter av typen "printer" innenfor en radius på femti meter. Dette avslører kontekst av typen "nære gjenstander" for eierartefakten.

10.2.1 Designanalyse i forhold til kravspesifikasjon

Jeg vil drøfte designanalysen av CIS og Stick-e notes som om de hører sammen i ett system. Selv om det ikke kommer klart frem i mine kilder om de faktisk gjør

det, vil jeg likevel anta det. Dette virker sannsynlig og fornuftig, da de begge omhandler det samme temaet, og begge er utarbeidet av de samme personene på samme sted (universitetet i Kent i Canterbury). Jeg vil benevne hele arkitekturen samlet under navnet CIS.

CIS tilfredsstiller krav 1 og 2 i kravspesifikasjonen. Ved hjelp av komponentene i sensor array kan rammeverket anvende både interne og eksterne sensorløsninger. I tillegg er det enkelt å legge nye sensorer inn i systemet. Dette kan gjøres uten å måtte endre på dets klientapplikasjoner eller artefakter samtidig. Dette sørger monitorene for. Disse dekker hvilke sensorer og synthesizers som anvendes for hver enkel artefakt, og er i stand til anvende dette dynamisk under eksekvering. Dette tillater at de beste sensorene til enhver tid benyttes. Krav 4 blir av den grunn innfridd.

CIS faller innenfor kategorien ”virkelig verden”: Den kan skille mellom flere entiteter ved hjelp av artifakt-komponentene. Imidlertid er det uvisst om disse kan ligge distribuert rundt på flere ulike maskiner. Jeg vil anta at klientapplikasjonene og sensorsystemene kan gjøre dette, eller i det minste artefaktene. I motsatt fall vil det bli mindre hensiktsmessig å snakke om flere artefakter i rammeverket, da systemet bare vil kunne anvende de som ligger lokalt. Ved hjelp av anchors, synthesizers og fri tilgang til alle artefakter, ligger uansett kommunikasjon av kontekst, krav 5, godt til rette.

CIS er også en av de få løsningene jeg har studert som aktivt benytter kontekst-historikk. Monitorene tar vare på artefaktenes tilstand i en logg, og anvender dette for å simulere forventede verdier. Krav 9 blir derfor innfridd. En interessant observasjon er imidlertid hvor i arkitekturen dette oppbevares. Dette er monitorenes ansvar. I Context Toolkit, som jeg vil drøfte senere, blir dette isteden gjort av sensorkomponentene selv.

Ved hjelp av synthesizer-komponentene innfris krav 10. Disse kan både omforme og utlede kontekst fra artefaktenes tilstand. Dessuten anvendes de likt som sensorene. CIS vurderer med andre ord sensorer og synthesizers som noenlunde samme abstraksjon; de leverer begge kontekstuell informasjon. Et resultat av dette er at monitorene kan knytte disse opp mot artefaktene på lik linje med sensorene. Dette kan igjen medføre at andre synthesizers plukker opp den utledede konteksten, og utleder den videre. Dermed realiseres også repeterende kontekst-utledning. Det spennende ved denne løsningen er hvordan utledningen inntreffer naturlig, uten spesielt tilpasset konstruksjon.

Også krav 11, abonnering og spørring etter kontekst, tilfredsstilles. Klientapplikasjonene har fri tilgang til alle artefakter i systemet, og dermed også til informasjon om deres kontekstuelle tilstander. Ved å skjule hvordan artefaktene får sin kontekst registrert, blir dem også enklere å inspisere fra klientapplikasjonene.

Ved hjelp av Stick-e notes adresserer rammeverket et system for å behandle kontekstrelevant informasjon. Denne teknologien er først og fremst en løsning for å tillate enkel informasjonsforfatting. Denne informasjonen har klientapplikasjonene også uavhengig tilgang til under eksekvering. Både krav 12, 17, og 20 blir derfor tilfredsstilt. Likevel adresserer ikke Stick-e notes noe system for å

uthente kontekstuelle opplysninger (krav 13), bare brukerinformasjon. Når dette kravet ikke tilfredsstilles, faller også hensikten med flere av de andre påfølgende kravene i kravspesifikasjonen bort. Likevel vurderer jeg teknologien som svært interessant. Jeg vil hevde at den danner et meget godt utgangspunkt for hvordan kontekstrelevant informasjon kan konstrueres og oppbevares.

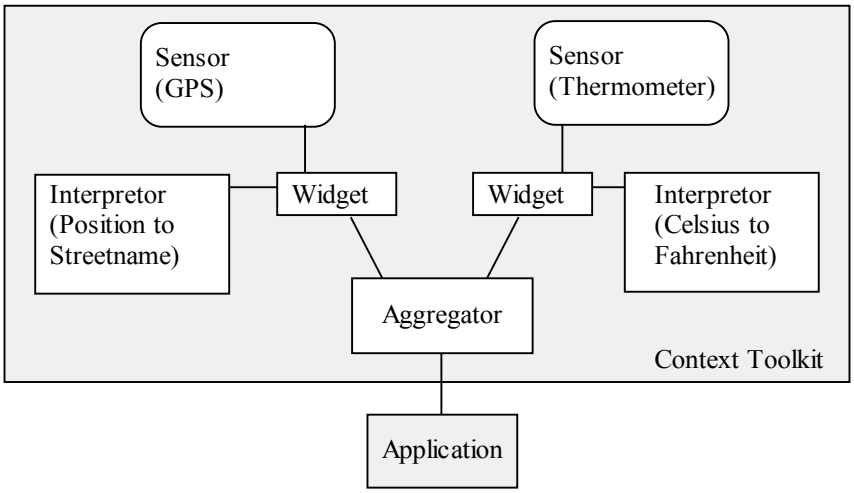
Krav som tilfredsstilles:	1–3, 5, 9–11, 12, 17, 19, 20
Krav som ikke tilfredsstilles:	4, 6–8, 13–16, 18

10.3 Context Toolkit

Som del av sin doktorgradsavhandling har Dey presentert et rammeverk for kontekstoppmerksomme applikasjoner benevnt Context Toolkit [Dey et al. 1999, Dey 2000]. Dey er inspirert av hvordan vanlig brukerinput blir registrert ved hjelp av utstyr som mus og tastatur, og hevder at dette har mange likheter med hvordan man også kan behandle sensorer: Kommunikasjon med den eksterne enheten må opprettes, og dataformatet den produserer må tilpasses. Begge deler bør gjøres transparent for applikasjonsutvikleren. Dey hevder imidlertid at sensorene skiller seg fra dette systemet ved ett sentralt punkt; de kan også være tilkoblet andre maskiner i nettverket enn kun den maskinen brukeren jobber på. Rammeverket må derfor tillate distribuert kommunikasjon mellom applikasjoner og sensorer.

Context Toolkit består hovedsakelig av fem komponenter: *Widget*, *Interpreter*, *Discoverer*, *Aggregator, og BaseObject* (se Figur 33). Blant disse er *Widget* mest sentral. Realiseringer av denne ”pakker” vekk kildekoden som kommuniserer med sensorene, og tilbyr et uniformt grensesnitt tilbake til klientapplikasjonene. I tillegg arver den fra klassen *BaseObject*. BaseObject er klassen som realiserer transparent distribuert kommunikasjon. Siden alle komponentene i løsningen arver denne, har også alle anledning til å kommunisere med hverandre over et nettverk. For å lete frem tilgjengelige komponenter anvendes *Discoverer*, som er løsningens svar på ”gule sider”.

Ofte kan Widget-komponentene levere kontekst på et annet format enn det en eller flere av klientapplikasjonene har behov for. En lokasjons-Widget kan for eksempel levere posisjoner i form av koordinater, mens man i en applikasjon isteden har behov for gateadresser. En slik omforming er *Interpretor*-komponentenes oppgave. En Interpretor kan benyttes av alle de andre komponentene i rammeverket, og omformer kontekst fra ett format til annet.



Figur 33: Context Toolkit – Figuren viser et eksempel på hvordan en konfigurering av rammeverket kan være sammensatt under eksekvering.

Aggregators ligner en del på Widgets ved at begge komponentene tilbyr kontekstuell informasjon. Forskjellen er at en Widget representerer én konkret sensor, mens en Aggregator kan ha ansvaret for en hel entitet. Dette realiseres ved at Aggregator-komponenten samler og kontrollerer flere Widgets, der hver og en av disse beskriver en bit av den samme entiteten. Aggregators løfter med andre ord abstraksjonsnivået ett hakk opp, fra sensorer til entiteter.

I en utvidelse av basisrammeverket, har Dey konstruert et overliggende lag benevnt *Situational Abstraction*. Fremfor å aksessere systemet via Widgets, Aggregators, og Interpretors, kan all kommunikasjon isteden skje via denne. Situational Abstraction er til for å la applikasjonene jobbe med kontekst i form av deklarative spesifikasjoner. Dette vil si at de kan fortelle rammeverket *hvilke* situasjoner de er interessert i, fremfor å avgjøre *hvordan* de må gå frem for å registrere dem. Dermed blir det isteden rammeverkets oppgave å finne frem til, og sette sammen, komponentene som bør anvendes. Klientapplikasjonene vil i så fall bare kjenne til situasjonene de skal respondere til, og ikke noen Widgets, Aggregators, eller Interpreters.

10.3.1 Designanalyse i forhold til kravspesifikasjonen

Context Toolkit løser mange av kravene fra kravspesifikasjonen. Først og fremst de fra 1-11, de som dreier seg om kontekstoppmerksomhet. Resten, anvendelse av kontekstrelevant informasjon, blir derimot ikke adressert. Dey hevder isteden at dette, i likhet med andre anvendelsesområder (jamfør 3.2), kan realiseres som et eget lag over rammeverket.

Løsningen kan vurderes som todelt. Man kan enten velge å la applikasjonene jobbe mot rammeverkskomponentene direkte, eller overlate kontrollen til Situa-

tional abstraction. Siden jeg vurderer det siste som mer i tråd med kravspesifikasjonen[1], vil jeg også drøfte med denne som utgangspunkt.

Jeg vil begynne med behandling av sensorer. Dette er kanskje rammeverkets største styrke. Både eksterne og interne sensorsystemer kan realiseres på en enkel måte ved hjelp av Widget-komponentene. Disse utgjør en grei abstraksjon i forhold til hvordan kontekst registreres, opp mot hvordan det anvendes. Ved å pakke sensorer inn i egne komponenter bak uniforme grensesnitt, blir blant annet gjenbruk av sensorløsninger enkelt. Dette har også vært et sentralt krav i Deys egen kravspesifikasjon; separering av sensorkomponenter fra applikasjoner. Bare i de tilfellene der man har behov for å anvende nye sensorer, må nye Widgets konstrueres og implementeres. Imidlertid går det nærmest automatisk å la eksisterende applikasjoner anvende nye sensorer. I de fleste tilfeller kan dette gjennomføres uten å måtte endre på applikasjonene. Særskilt gjelder dette ved anvendelse av Situational abstraction. Gjennom denne komponenten vil applikasjonene aldri måtte ha noe forhold til hvilke sensorer som bør benyttes. Oppsummert vil jeg derfor hevde at både krav 1 og 2 løses meget tilfredsstillende.

Når det gjelder krav 4, dynamisk skifte av sensorsystemer, er situasjonen litt annerledes. Situational abstraction er i stand til å skifte mellom ulike sensorløsninger etter hvert som de kobles til eller fra. Imidlertid blir ingen av tjenestekvalitetsattributtene, spesifisert i krav 4, involvert. En av farene med dette, som Dey selv påpeker, er problemet med dekning: En Widget som hevder at den kan produsere verdier utover det den egentlig kan. Situational Abstraction kan derfor bli ”lurt” til å anvende en sensor som ikke kan levere gyldige kontekstverdier for brukeren. Jeg vurderer av den grunn krav 4 som halvveis innfridd.

Context Toolkit tilhører kategorien ”virkelig verden”. Ved hjelp av Aggregators kan klientapplikasjonene skille mellom flere entiteter i systemet. Dette gjør at de også enkelt kan inspisere konteksten til alle brukerne under eksekvering. Ved hjelp av tilgang til alle Aggregators i systemet, kan de uthente sosial kontekst for hver enkelt. Krav 5 blir derfor støttet.

Selv om manuell kontekstspesifikasjon, krav 7, ikke har fått noen direkte støtte i rammeverket, kan dette løses implisitt. Først vil jeg likevel påpeke at i de tilfeller hvor man ikke benytter Situational Abstraction, så vil dette behovet falle bort. Når klientapplikasjoner jobber mot sensorer (Widgets) direkte, så kan de også manipulere deres verdier som det passer. I motsatt tilfelle, med Situational Abstraction, vil imidlertid manuell kontekstspesifikasjon danne mening. Her vil jeg også hevde at en slik spesifikasjon kan tillates ved å konstruere egne Widgetkomponenter spesielt tiltenkt dette formålet. Disse kan ha direkte kontakt med klientapplikasjonene, og sørge for at brukernes egne spesifikasjoner blir gjort tilgjengelig for rammeverket. Et annet spørsmål er hvorvidt man kan overstyre de kontekstleverandørene som allerede finnes i systemet fra før (andre Widgets). Om man anvender Situational abstraction, vil ikke denne favorisere kontekst som

[1] Kravspesifikasjonen har en gjennomgående tilbøyelighet for at behandlingen av kontekstoppmerksomhet og kontekstrelevant informasjon skal foregå skjult. Det vil si at minst mulig av dette skal overlates til klientapplikasjonsutviklerne.

kommer fra brukeren. Dermed vil den heller ikke være i stand til å konsekvent overstyre andre leverandører når dette er ønskelig. Dermed løses heller ikke krav 8 tilfredsstillende.

En av oppgavene som er lagt til Widgets, er å oppbevare konteksten de produserer i en historikklogg. Dette gjør de hver for seg, uavhengig av de andre komponentene. Utviklere står derfor fritt til å avgjøre hvordan dette skal ivaretas hos hver enkelt Widget. Grunnen til at historikkansvaret er delegert til disse, er fordi Widget-ene er *autonome*: De kan eksekvere selvstendig, uavhengig av applikasjonene som anvender de. Jeg vil derfor konkludere med at krav 9 løses tilfredsstillende. Imidlertid forutsetter dette at Situational abstraction er i stand til å benytte historikkinformasjonen fra Widget-ene som kontekstuell informasjon. Det vil si, klientapplikasjonene kan spesifisere situasjoner som tvinger Situational Abstraction til å analysere historikkloggen til en eller flere Widgets for å kunne avgjøre om situasjonen har inntruffet.

Context Interpretors het komponentene i rammeverket som omformet kontekst fra ett format til et annet. Disse var i stand til å ta imot kontekstparametere som input, og utlede ny kontekst tilbake. Disse kan anvendes av applikasjoner, Widgets, Aggregators, eller andre komponenter både i og utenfor rammeverket. Dette inkluderer Situational abstraction. Situational abstraction vil analysere situasjonene som blir gitt fra applikasjonene, vurdere hvilke Widgets, Interpretors, og Aggregators som er tilgjengelig for å løse oppgaven, og foreta abonneringer på disse. Hele krav 10 blir derfor innfridd.

Det siste kravet jeg vil drøfte, krav 11, spesifiserer at klientapplikasjonene bør ha tilgang til å inspisere brukernes kontekst. Akkurat dette er imidlertid Context Toolkit konstruert for. Siden rammeverket kun behandler kontekstoppmerksomhet, følger dette naturlig. Hva kontekstoppmerksomheten skal anvendes til blir overlatt til klientapplikasjonene. Slik sett er klientapplikasjonene nødt til å ha denne tilgangen, i motsatt fall vil oppmerksomheten rammeverket tilbyr aldri bli benyttet til noen som helst.

Krav som tilfredsstilles:	1–3, 5, 7a, 8a, 9, 10, 11
Krav som ikke tilfredsstilles:	4, 7b, 8b, 12–20

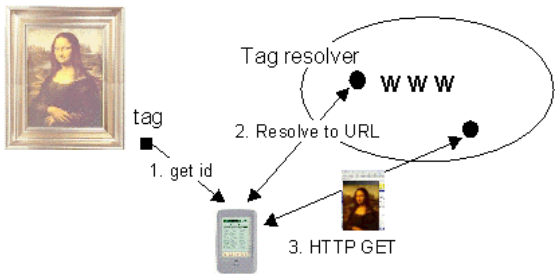
10.4 CoolTown

Jeg vil til sist drøfte CoolTown fra Hewlett-Packard [Kindberg et al. 2001]. Denne løsningen baserer seg på anvendelse av kontekstrelevant informasjon ved å assosiere internettsider til fysiske gjenstander. Kindberg hevder følgende:

”Currently the Web is largely a virtual space: a space of websites, online malls, and chatrooms. These virtual locations have little correspondence with physical spaces. While much of the information on the Web describes the world we physically inhabit, there are few systematic linkages to real world entities. This is unfortunate, because most of our activities concern physical objects other than computers.”

– [Kindberg et al. 2001, side 1]

For å knytte objekter i den fysiske verdenen opp mot Internett, anvendes *beacons*. Beacons er små trådløse sendere som sender infrarøde signaler. Disse festes i nærheten av de ønskede objektene, og kringkaster med jevne mellomrom en av to verdier; enten de ønskede objektenes assosierte URL, eller en tilsvarende id. Disse kan plukkes opp av PDA-ene i nærheten, og tillater nedlasting av deres assosierte nettsider over Internett (ved hjelp av et trådløst nettverk). Om signalet var en id-verdi (benevnt ”indirect sensing”), omformes disse først til en URL ved hjelp av en *Tag resolver* før nedlastingen kan begynne (se Figur 34).



Figur 34: CoolTown – Illustrasjonen viser hvordan systemet henter frem en lokasjonsrelevant Internettside ved hjelp av *indirect sensing*. (Illustrasjonen er hentet fra [Kindberg et al. 2001]).

Objekter i CoolTown blir benevnt som *ting*. Maleriet i Figur 34 er et eksempel på en slik ”ting”. Sammen med *mennesker* og *steder* utgjør disse til sammen hovedabstraksjonene i løsningen. Disse danner byggeklossene for å modellerer den fysiske verdenen. Av den grunn vurderer jeg også CoolTown til å falle innunder kategorien ”virkelig verden”.

Et ”sted” blir representert i CoolTown ved hjelp av en *PlaceManager*. En PlaceManager er en slags proxyserver som ligger utenpå vanlige webservere og kontrollerer adgangen til deres dokumenter. Dokumentene, igjen, representerer ulike ”ting”. Et sted er med andre ord ikke representert som en egen internettside, men isteden som en applikasjon som samler og kontrollerer adgangen til flere relaterte *ting*.

Menneskene blir til sist tilgjengeliggjort i CoolTown over tjenesten *Weblink*. Weblink holder rede på hvor alle til enhver tid befinner seg, og presenterer dette

på hver enkelts personlige ”hjemmeside”. Disse sidene gjør at man enkelt kan finne hverandre rent fysisk, eller alternativt, diskutere med hverandre over egne diskusjonsfora implementert på de samme sidene.

10.4.1 Designanalyse i forhold til kravspesifikasjon

CoolTown er ikke i første rekke et rammeverk for behandling av kontekstoppmerksomhet. Systemet skisserer blant annet ingen generell struktur for anvendelse av sensorer. Jeg oppfatter den heller som en applikasjon. En applikasjon på lik linje med vanlige nettlesere, men utvidet. Utvidet for å kunne knytte internettsider opp mot gjenstander i den fysiske verdenen. Dette er mye likt slik The Lancaster Guide også var realisert; her ble vanlige internettsider anvendt som kontekstrelevant informasjon (jamfør 4.4). Imidlertid har CoolTown et relativt generelt anvendelsesområde. Den kan tilby en bred kategori med tjenester. Alt fra bankvirksomhet til turistguider. En Internettside kan for eksempel både inneholde informasjon i form av ren tekst, eller en applikasjon i form av en Java-Applet. Det siste tillater, mer spesielt, konstruksjon av *kontekstrelevante applikasjoner*: Applikasjoner som bare er gyldige (relevante å benytte) når brukeren befinner seg innenfor dens aktiveringskontekst.

Ved å anvende Internett slik CoolTown gjør, får man også med mange av dets fordeler: Enkel forfatting og distribusjon [Brown 1995]. Tilgang til informasjon fra mange ulike maskiner og operativsystem [Kindberg et al. 2001]. Og ikke minst, en godt innarbeidet og programmeringsspråkuavhengig standard for presentasjon av informasjon og komplekse brukergrensesnitt.

Rammeverket adresserte ikke noen generell struktur for behandling av kontekst. Kun lokasjonsoppmerksomhet ved hjelp av beacons var implementert. Dermed er det heller ikke noe grunnlag for å drøfte punktene i kravspesifikasjonen som angår dette, krav 1 til 11. Derimot vurderer jeg løsningen som særdeles interessant når det gjelder behandlingen av kontekstrelevant informasjon, krav 12. Dette blir løst på en smart måte. Spesielt krav 14 innfris nærmest gratis; dette var mulighet for å konstruere dynamiske dokumenter. Slik Internett er i dag, er dette allerede en svært utbredt funksjonalitet. Dynamisk generering av nettsider kan gjøres både på serversiden[1] og klientsiden (dynamisk html) i internettarkitekturen. CoolTown anvender denne muligheten selv i Weblink. Her har hver person sin egen hjemmeside som kontinuerlig oppdateres i forhold til deres lokasjon. Hvorvidt systemet tar vare på en informasjonshistorikk over tid (krav 15) blir isteden opp til hver enkelt applikasjon på serversiden å implementere.

Når det gjelder kravene 17 til 20, blir drøfting av disse som drøftingen over; mer et spørsmål om hvordan Internett fungerer enn hvordan CoolTown fungerer. CoolTown blir, som nevnt, bygd med Internett som infrastruktur. Siden man der

[1] På serversiden har blant andre Sun og Microsoft levert relativt store programmeringsmiljøer for å realisere dette. Mer om slike løsninger kan leses hos: Sun: http://java.sun.com/j2ee/ (11.04.02), eller Microsoft: http://msdn.microsoft.com/ (11.04.02).

har man har tilgang til å både legge til, og endre, eksisterende HTML-dokumenter under eksekvering (krav 17), får man også mulighet til dette i CoolTown. Videre har Internett et godt innarbeidet system for å både distribuere og begrense tilgangen til informasjon (krav 18). Ved å aksessere dokumentene fra vanlige nettlesere, har man også uavhengig tilgjengelighet til informasjon i forhold til brukerens kontekst (krav 19). Oppsummert vurderer jeg derfor CoolTown som en spennende, relevant, og elegant arkitektur.

Krav som tilfredsstilles:	12, 14, 17–20
Krav som ikke tilfredsstilles:	1–11, 13, 15–16

10.5 Oppsummering

Jeg har i dette kapitlet gjennomgått tre eksterne rammeverk. Jeg har presentert deres konstruksjon, og analysert dem opp mot kravspesifikasjonen. Hvilke krav som ble besvart av hver enkelt, varierte. Mens Context Toolkit fokuserte på kontekstoppmerksomhet, vektla CoolTown anvendelse av kontekstrelevant informasjon. Kun en av løsningene, CIS, adresserte begge områdene mer utfyllende.

Rammeverkene jeg presenterte, var de jeg fant til å være mest relevante for oppgavens tema[1]. Selv om disse besvarte flere av kravene fra kravspesifikasjonen hver for seg, var likevel ingen i nærheten av å løse alle. Av den grunn vil jeg også anta at det sannsynligvis ikke eksisterer noen rammeverk pr. i dag som kan støtte utvikling av applikasjonene jeg har studert, fullt ut. Imidlertid har dette studiet gitt resultater i form av ideer og inspirasjon rundt design. Alle løsningene ble, som nevnt, også analysert for dette henseende. Disse resultatene vil jeg ta med meg under konstruksjonen av Condor.

[1] Jeg har også sett på Context Fabric [Hong 2000], og en arkitektur av Efstratiou [Efstratiou et al. 2001]. Efstratiou er forøvrig en av personene bak The Lancaster Guide.

Kapittel 11

Løsningsforslag: Condor

Etter å drøftet sentrale begreper innen oppgavens område, redegjort for både eksterne og egne eksempelapplikasjoner, drøftet relevant funksjonalitet hos disse, sammenfattet resultatene herfra i en kravspesifikasjon, og drøftet eksterne rammeverk opp mot denne, har jeg nå kommet frem til mitt eget løsningsforslag. Dette har jeg valgt å gi navnet Condor (Context Document Framework).

Under utviklingen av Condor har jeg hentet en del inspirasjon fra de eksterne rammeverkene som ble presentert i forrige Kapittel. Dette var også noe av hensikten med analysen av dem. Likevel divergerer denne løsningen ved ett sentralt punkt. Fremfor å skille behandlingen av kontekstrelevant informasjon ut fra behandlingen av kontekstoppmerksomhet, vil jeg forsøke å smelte dette sammen til ett lag. Bakgrunnen for dette vil jeg drøfte under.

Jeg vil begynne neste avsnittet med å drøfte en viktig observasjon. Denne vil danne et slags utgangspunkt for Condors arkitektur. I tillegg vil den bringe meg over fra observasjon til idé. Her vil jeg presentere et overordnet forslag på hvordan jeg vil gå frem for å realisere rammeverket. Deretter vil jeg redegjøre for Condors konstruksjon mer detaljert. Jeg vil ikke gå helt ned på kildekodenivå, men fremvise og drøfte UML-klassediagrammer. Til sist vil jeg drøfte Condor som en helhet opp mot kravspesifikasjonen fremsatt i Kapittel 9.

Før jeg begynner vil jeg presisere at Condor i løpet av dette studiet aldri har blitt implementert. Derfor er det heller aldri blitt utprøvd under reell applikasjonsutvikling. På grunn av dette kan man heller ikke forvente at en endelig versjon av Condor vil fremstå nøyaktig slik som det presenteres her. Isteden fortsetter rammeverk mer generelt å utvikle seg i lang tid etter en initsiell konstruksjon [Fayad et al. 1999] (jamfør Kapittel 2). Det er med andre ord høyst naturlig å vurdere Condor som et utgangspunkt for et rammeverk, fremfor en endelig versjon.

11.1 En viktig observasjon

Igjennom oppgaven har jeg behandlet kontekst og kontekstrelevant informasjon separat, som om det var to atskilte konsepter. I Kapittel 3.2 drøftet jeg hvordan kontekstrelevant informasjon kunne vurderes som et anvendelsesområdene for kontekstoppmerksomhet. Jeg delte også identifikasjonen av funksjonalitet inn i to atskilte kapitler: Anvendelse av kontekstoppmerksomhet i Kapittel 7, og Anvendelse av kontekstrelevant informasjon i Kapittel 8. Det samme var også tilfellet med kravspesifikasjonen. I tillegg vil jeg argumentere for at rammeverkene jeg studerte i Kapittel 10 fulgte samme logikk. De fleste av disse anvendte enten kontekstoppmerksomhet eller kontekstrelevant informasjon. Kun ett av dem, CIS, belyste begge. Dey argumenterer for at "tagging" av kontekst til informasjon kan ligge som et tjenestelag ovenfor kontekstregistreringen [Dey 2000]. I Condor vil jeg forsøke å endre litt på dette synet.

Jeg vil begynne med en viktig observasjon. Ofte kan det være vanskelig å skille mellom hva som i en situasjon er kontekst, og hva som isteden er dens innhold. Dette avhenger av hva hver enkelt observatør ønsker å se. I Field assistant besto for eksempel en observasjon av tid, sted, og dyreslag [Pascoe 1998]. Hva som av dette ble vurdert som innhold eller kontekst, var avhengig av hvordan det skulle anvendes. I noen sammenhenger kunne for eksempel dyreslaget være informasjonen, mens tid og sted ble dens kontekst. I andre sammenhenger kunne dyreslag og tidspunkt vurderes som kontekst, mens stedet for registreringen var informasjonen [Pascoe 1998]. Skillet mellom hva som var kontekst og kontekstrelevant informasjon ble derfor vag. Jeg vil følge denne problematikken litt videre.

Sammenblandingen av kontekst og kontekstrelevant informasjon kom også til syne under kravanalysen. I Kapittel 8 drøftet jeg blant annet hvordan kontekstrelevant informasjon også kunne inneholde mange kontekstuelle opplysninger. Jeg argumenterte på grunnlag av dette at slik informasjon kunne vurderes til å inneholde to felter; ett ment for brukeren (brukerinformasjonen) og ett tiltenkt applikasjonen (postkonteksten). Dermed ville kontekstrelevant informasjon i noen situasjoner også bidra med kontekst. Dette forutsatte selvsagt at applikasjonene var i stand til å trekke ut slike opplysningene under registreringsprosessen.

Jeg argumenterte også for det motsatte. I Kapittel 7.2.2 gjorde jeg rede for hvordan kontekst i mange tilfeller også kunne tjene som kontekstrelevant informasjon. Nære steder var et eksempel på dette. Et nært sted, som for eksempel en bussholdeplass, kunne både opptre som kontekstuell informasjon og kontekstrelevant informasjon. I det siste tilfellet ble informasjonen kontekstrelevant da den var direkte knyttet til brukerens lokasjon. Til sammen har dette forsterket inntrykket av at kontekst og kontekstrelevant informasjon kan oppfattes som to sider av samme sak.

Når kontekstrelevant informasjon bidrar med kontekst, oppstår en spennende situasjon. Postkonteksten som spesifiseres får både en aktiveringskontekst og en deaktiveringskontekst. Dette er egenskapene til kontekstrelevant informasjon

(jamfør 3.3). Kontekstbidraget blir med andre ord ikke gyldig for enhver situasjon en bruker befinner seg i. Han må befinne seg innenfor dens aktiveringskontekst om informasjonen skal beskrive hans situasjon. Dette er helt naturlig. Man kan for eksempel se for seg en annotering i Mocado festet til en statue. Denne inneholder to typer informasjon; en til brukeren (brukerinformasjonen), og en til applikasjonen (postkonteksten). Det siste forteller applikasjonen at brukeren er i nærheten av en statue (fysisk kontekst). Imidlertid vil dette bare være gyldig i de tilfeller hvor han befinner seg innenfor dens aktiveringskontekst.

Et interessant spørsmål gjør seg gjeldene her. Gjelder betinget anvendelse av kontekstuell informasjon bare kontekst som registreres fra kontekstrelevant informasjon? Eller vil det også gjelde kontekst utvunnet fra andre registreringsmetoder (jamfør 7.1)? Har for eksempel en måling som leveres fra et termometer både en aktiveringskontekst og en deaktiveringskontekst? Jeg mener at svaret på dette er ”ja”. Dette spørsmålet ble også delvis drøftet i Kapittel 7.1.1. Der gjorde jeg rede for hvordan man har behov for å la applikasjoner bytte mellom ulike sensorløsninger dynamisk i forhold til en rekke tjenestekvalitetskrav. Spesielt blant disse var dekning. Dekning beskrev rammene for hvilke observasjoner en sensor kunne levere. Et termometer som hang utenfor et butikkvindu, eksemplifiserte jeg, kunne bare levere gyldige verdier til de som befant seg like i nærheten. Observasjonene den produserte hadde en bestemt gyldighetsradius (dekningsområde), nemlig en aktiveringskontekst. Dersom en bruker kom innenfor denne konteksten, ble temperaturmålingene gjort gjeldene også for han. Analogt ville de også deaktiveres om han forflyttet seg videre inn i butikken; temperaturen inne ville jo høyst sannsynlig være annerledes enn den utenfor. Observasjonene ble i så fall ugyldige. Med andre ord leverte sensoren bare gyldige observasjoner for brukeren når han befant seg i nærheten av termometeret.

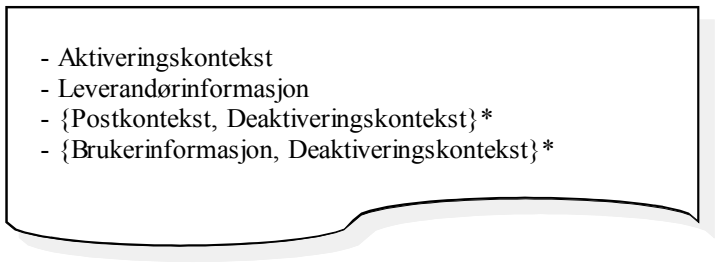
Som en følge av det jeg har drøftet over, vil jeg argumentere for at behandling av kontekstrelevant informasjon ikke bør skilles ut fra behandlingen av kontekstoppmerksomhet. Kontekstrelevant informasjon bør isteden delta som en registreringsmetode for kontekst. Disse områdene bør, av den grunn, heller integreres tettere sammen.

11.2 Fra observasjon til idé

For å integrere anvendelse av kontekstoppmerksomhet med anvendelse av kontekstrelevant informasjon, vil jeg ta utgangspunkt i teknologien Stick-e notes fra Brown (jamfør 10.2). Her ble kontekst og kontekstrelevant informasjon assosiert og sammenfattet i separate tekstfiler. I Condor vil jeg gjøre bruk av et tilsvarende konsept som jeg vil kalle *kontekstdokumenter*. I utgangspunktet er kontekstdokumentene ganske like Stick-e notes-ene. Imidlertid kan kontekstdokumentene vurderes som en videreutvikling av Stick-e notes konseptet.

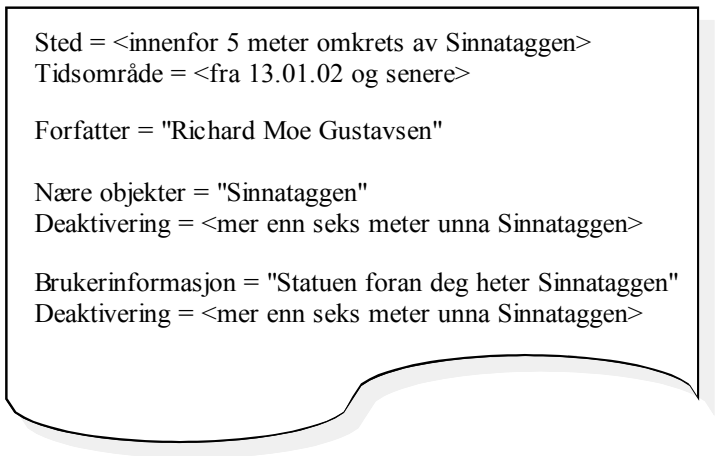
Oppsummert vil et kontekstdokument i Condor inneholde følgende tre felter: en aktiveringskontekst, en brukerinformasjon, og en postkontekst. I forhold til

Stick-e notes er postkonteksten ny. Både brukerinformasjonen og postkonteksten kan videre bestå av flere mengder med informasjon, hvorav hver enkelt mengde også vil ha en egen separat deaktiveringskontekst (se Figur 35). I tillegg vil jeg introdusere et nytt felt som ikke er blitt omtalt i oppgaven tidligere: *Leverandør-informasjonen*. Denne beskriver hvem som har produsert dokumentet, tidspunkt for produksjonen, og informasjonens assosierte tjenestekvalitet. Dette er hendig informasjon for klientapplikasjonene som ønsker å benytte dokumentene senere. Mer om Leverandørinformasjonen kommer nedenfor.

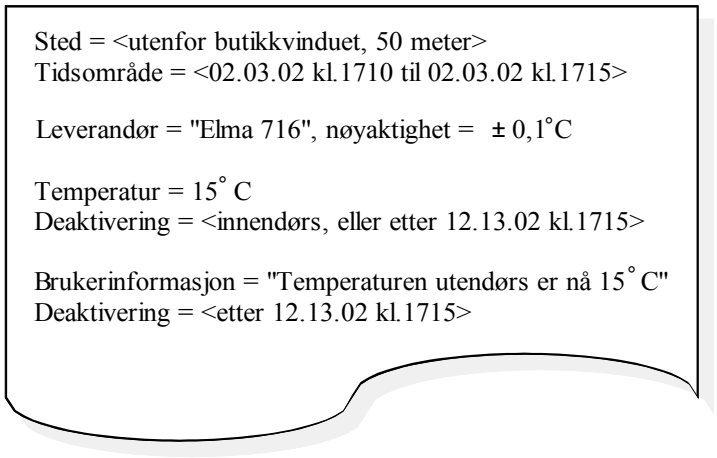


Figur 35: Et kontekstdokument – En generell beskrivelse av et kontekstdokument. Stjernen betyr at innholdet i krøllparentesene kan forekomme null eller flere ganger i dokumentet.

Alle komponenter som skal publisere kontekstuell-, eller kontekstrelevant informasjon i Condor, gjør dette ved hjelp av å produsere og distribuere kontekstdokumenter. Figur 36 viser et overordnet eksempel på hvordan for eksempel en annotering i Mocado kunne blitt spesifisert ved hjelp av et kontekstdokument. Det samme er tilfellet for Figur 37. Her er imidlertid leverandøren et termometer. Slike sensorer (som et termometer er) vil publisere et nytt dokument for hver eneste observasjon de foretar.

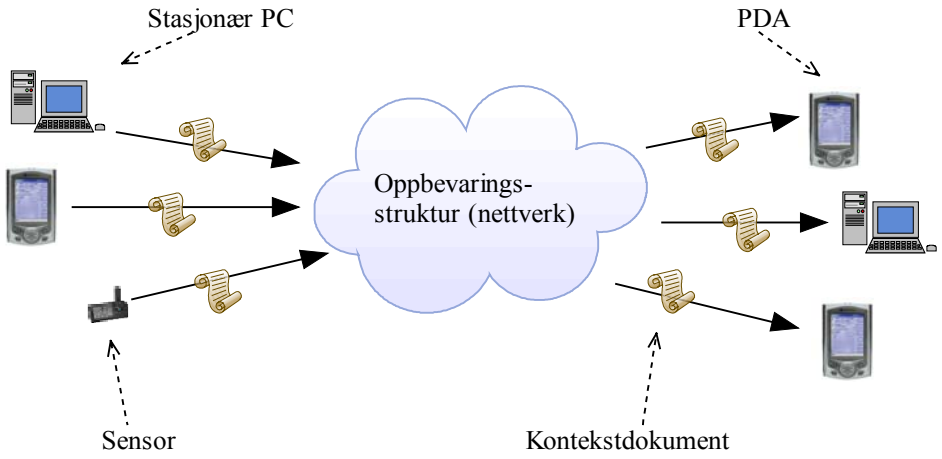


Figur 36: En annotering – Kontekstdokumentet viser (ved hjelp av pseudokode) et eksempel på en annotering fra Mocado. Selv om deaktiveringskontekstene i dette tilfellet er like, kan de likevel variere i andre tilfeller.



Figur 37: En sensorobservasjon – Kontekstdokumentet viser (ved hjelp av pseudokode) et eksempel på hvordan en sensorkomponent kunne ha levert en observasjon til rammeverket. Sensoren i dette tilfellet er et termometer. Nøyaktigheten som spesifiseres i et dokumentet vil kun gjelde observasjonen dokumentet beskriver.

Alle dokumenter som publiseres, oppbevares i en distribuert oppbevaringsstruktur. Her lagres de i henhold til deres lokasjon og *gyldighetstid*. Det siste spesifiserer hvilket tidsintervall dokumentet skal oppfattes som gyldig, og fastsettes i aktiveringskonteksten av leverandørkomponenten selv. For en annotering vil dette vanligvis gjelde fra dokumentets produksjonstid og fremover, mens den for en observasjon bare kan inkludere noen få sekunder etter den ble foretatt. Etter dette, skal den ikke lenger oppfattes som relevant. Uansett skal alle publiserte dokumenter oppbevares så langt det lar seg gjøre (med tanke på lagringskapasitet). På den måten kan enhver klientapplikasjon tilknyttet Condor hente frem informasjon for situasjoner bakover i tid. Dette tillater situasjonssimulering. Alle som ønsker å søke etter publiserte dokumenter, henvender seg med andre ord til oppbevaringsstrukturen (se Figur 38). Ved å oppbevare hele dokumenter her, oppnår man også en meget viktig egenskap. Strukturen lagrer ikke bare sensorenes observasjonsverdier, men også deres aktiverings-, og deaktiveringskontekst (samt assosiert tjenestekvalitet, som spesifisert i leverandørinformasjonen). Dette er viktig om sensoren eller annoteringen for eksempel flytter på seg. Da blir en ren verdilogg utilstrekkelig om situasjonssimuleringer bakover i tid skulle kunne la seg gjennomføre.



Figur 38: Overordnet oversikt over Condor – Sensorer, PDA-er, eller andre maskiner/applikasjoner kan publisere kontekstdokumenter til den distribuerte oppbevaringsstrukturen. På samme vis kan disse dokumentene hentes ut igjen etter behov (og situasjon).

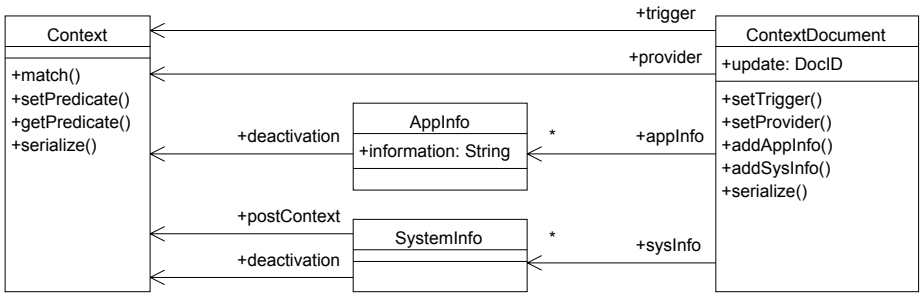
11.3 Fra idé til konstruksjon

I forrige avsnitt gjorde jeg rede for hvordan kontekst og kontekstrelevant informasjon kunne generaliseres og sammenfattes i kontekstdokumenter. I tillegg viste jeg en meget overordnet modell for hvordan slik informasjon kan utveksles i Condor. Fra nå av vil jeg benevne alle som leverer eller søker etter kontekstdokumenter for *klienter*. De er klienter i forhold til oppbevaringsstrukturen. Den delen av rammeverket som er tiltenkt klientene, vil jeg videre benevne som *klientsiden*, mens oppbevaringsstrukturen isteden benevnes *serversiden*.

Klientsiden kan vurderes som den viktigste delen for klientapplikasjonsutviklerne å kjenne til. Om man ønsker å legge til nye sensorer, eller anvende kontekstrelevant informasjon, gjøres dette herfra. En realisering av klientsidearkitekturen må derfor ligge implementert hos hver klient. Når et dokument overføres mellom klient-, og serverside, er formatet på dokumentet serialisert ned til ASCII-tekst (i likhet med dokumentene i Figur 36 og Figur 37, og i henhold til Stick-e notes arkitekturen forøvrig[1]). Imidlertid oppbevares de ikke slik så lenge de befinner seg på klientsiden. Dette ville blitt uhåndterlig. Isteden bevares de i en egen struktur, avbildet i Figur 39. Diagrammet viser hvordan klassen ContextDocument (som representerer et kontekstdokument) oppbevarer den drøftede dokumentinformasjonen ved instansieringer[2] (jamfør 11.2).

[1] I denne oppgaven vil jeg ikke behandle hvordan slike dokumenter kan representeres ved hjelp av ASCII-tekst. Likevel vurderer jeg XML som en fornuftig teknologi for å strukturere slik informasjon.

[2] Se Figur 43 for en full oversikt over Condors klientarkitektur.



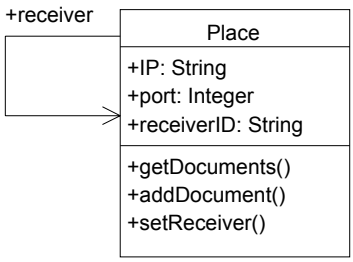
Figur 39: Kontekst og kontekstdokument – Condors klientarkitektur med fokus på ContextDocument og Context.

I figuren tilsvarer assosiasjonene *trigger*, *provider*, *appInfo*, og *sysInfo* henholdsvis aktiveringskontekst, leverandørinformasjon, brukerinformasjon, og postkontekst. Begge de to siste kan bestå av flere informasjonsbiter (representert med klassene *AppInfo* og *SysInfo*[1]), og har hver for seg sine egne deaktiveringskontekster. Til sist har også *ContextDocument* en variabel benevnt *update*. Denne forteller om en instanse av et publiseringsklart dokument er en oppdatering av et annet. I så fall skal serversiden "lukke" igjen tidsintervallet hos det gamle dokumentet før det nye legges til. Dette vil hindre at en annen klient i ettertid aktiverer flere versjoner av samme dokument (både utdaterte og aktuelle). Hvilken identitet et dokument får, bestemmes av serversiden når dokumentet mottas, og leveres tilbake til klienten som svar på publiseringen.

På venstre side av diagrammet vises klassen *Context*. Denne er satt til å oppbevare kontekstpredikater for en rekke andre klasser i rammeverket. Spesielt inneholder denne klassen metoden *match*, som kan avgjøre om en annen Context-instanse faller innenfor dens egen spesifikasjon. I så fall vil den andre instansen vurderes som gyldig i forhold til denne. Både Context og ContextDocument inneholder også metoden *serialize*. Denne benyttes for å serialisere objektene ned til et reint ASCII-format, noe som gjør at de begge kan overføres fritt mellom klient-, og oppbevaringsstruktur. Tilsvarende har begge også en konstruktør for å bygge de opp igjen fra samme format.

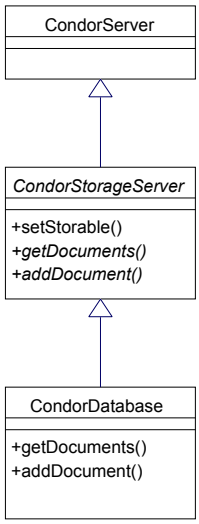
For å sende et dokument fra en klient til en oppbevaringsstruktur, anvendes en instanse av klassen *Place* (se Figur 40). Denne inneholder først og fremst en IP-adresse og et portnummer, og henviser vanligvis til en lagringsserver i nettverket. Alle klienter som produserer dokumenter har med andre ord et forhold til denne. For å publisere et dokument, trenger man bare å opprette en instanse av ContextDocument, fylle inn dens verdier, og sende den til metoden *addDocument* her. Denne sørger for at dokumentet blir serialisert og sendt til riktig mottaker.

[1] Disse to navnene er forkortelser for ApplicationInformation og SystemInformation. Det første er informasjon som skal leveres til klientapplikasjonene (brukerinformasjonen), mens det andre er postkontekst til systemet (Condor).



Figur 40: Klassen Place – Alle komponenter i Condor som skal sende eller søke etter dokumenter anvender denne som grensesnitt til oppbevaringsstrukturen.

På andre siden av Condor, oppbevaringssiden, mottas tekstdokumentet ved hjelp av klassen *CondorServer* (se Figur 41). Denne er relativt generell, og lytter etter innkomne meldinger for alle typer nettverkskomponenter i løsningen. En spesialisering av denne, *CondorStorageServer*, behandler det mottatte dokumentet mer utfyllende, og bygger det opp til en instanse av ContextDocument. Mer om hvordan CondorStorageServer fungerer vil jeg imidlertid beskrive senere. Til sist spesialiserer *CondorDatabase* CondorStorageServer igjen, og sørger for at dokumentet blir oppbevart i en eller annen database, tilgjengelig for andre klienter.



Figur 41: Et utsnitt av Condors oppbevaringsstruktur – Dokumenter mottas av CondorDatabase som er en spesialisering av CondorStorageServer. Her blir de oppbevart som kontekstshistorikk.

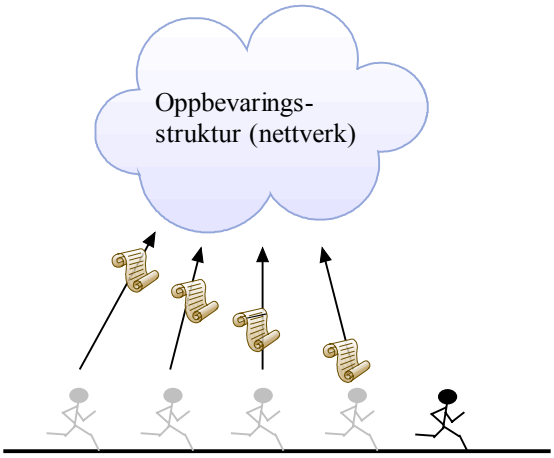
Tilsvarende som for å sende dokumenter, anvendes også Place-instanser for å søke etter dokumenter. I slike tilfeller benyttes en instanse av klassen Context som søkefilter. Denne sendes til metoden *getDocuments* i klassen Place. Derfra vil forespørselen overføres til oppbevaringsstrukturen og prosesseres videre der. Imidlertid vil ikke svaret som returneres (dokumentene) komme tilbake samme vei. Place-instanser anvendes bare for å sende asynkrone forespørsler og dokumenter en vei. Den mottar aldri noe tilbake. Isteden må hver klient anvende en

egen CondorServer benevnt *World*. Denne er konstruert spesielt for å motta svar-dokumenter tilbake fra serversiden.

Frem til nå har jeg beskrevet de mest generelle komponentene av rammeverket. De som tar seg av å representere, sende, og spørre etter kontekstdokumenter. Disse tilhører grunnelementene i løsningen. Ved hjelp av disse kan man konstruere og anvende flere ulike typer klienter (som blant annet sensorsystemer, noe jeg kommer jeg til senere). Likevel mangler en viktig abstraksjon, artefaktene. I Context Information Service (CIS, jamfør 10.2) ble disse anvendt for å modellere den fysiske verdenen [Pascoe 1998]. Artefaktene kunne representere personer, steder, ting, eller lignende. I tillegg hadde de attributter som navn, typebestemmelse, og en kontekstilstand. I Condor har jeg valgt å kalle tilsvarende abstraksjon for entiteter (se Figur 43). En entitet er strengt tatt ikke noe mer enn en komponent som både konstruerer, sender, og spør etter kontekstdokumenter. På den ene siden publiserer den informasjon om seg selv, som lokasjon, identitet, aktivitet, og lignende. På den andre siden søker den etter relevant informasjon i henhold til sin egen kontekstuelle situasjon.

Før jeg går videre med entitetene, vil jeg gjøre oppmerksom på et viktig formål bak Condors oppbevaringsstruktur. Fremfor at en realisering av denne bare vil bestå av én lagringsserver, kan den isteden bestå av mange. Disse kan til og med danne et hierarki. Enda viktigere er det at disse kan danne utgangspunktet for flere virtuelle verdener. Mens noen ”verdener” bare vil innehold applikasjonsspesifikke dokumenter (som kun én spesiell applikasjon vil anvende), er andre tiltenkt de mer generelle og applikasjonsuavhengige. Spesielt blant disse er den som modellerer den fysiske verdenen (fra nå av kalt den fysisk-virtuelle verdenen). Denne vil inneholde informasjon om steder, gjenstander, entiteter, og lignende, fysiske, objekter (i form av kontekstdokumenter). Dette er informasjon som kan være interessant å anvende på tvers av ulike klientapplikasjoner. Av den grunn vil den også være tilgjengelig for alle slike applikasjoner. Det er også til denne verdenen de *fysiske entitetene* vil publisere informasjon om seg selv. Med fysiske entiteter menes entiteter som simulerer fysiske abstraksjoner (som for eksempel en bruker). Mer applikasjonsspesifikke dokumenter (som et dokument som minner deg på å fylle bensin på bilen når tanken nærmer seg tom) kan isteden plasseres i andre, lokasjonsuavhengige, verdener. Entitetene står uansett fritt til å søke etter dokumenter fra flere forskjellige oppkonstruerte verdener. Hvilke som skal anvendes blir opp til applikasjonsutvikleren. Jeg vil komme mer tilbake til serversiden og dens oppbevaringsstruktur senere.

En hovedidé bak entitetene er at de skal ”leve” på egen hånd. Straks en klientapplikasjon har konstruert en entitet, vil den begynne å produsere og distribuere kontekstdokumenter med jevne mellomrom (se Figur 42). På samme vis vil den også lete frem relevante dokumenter og sende dem ned til klientapplikasjonen. For applikasjoner som Mocado, kan entiteten for eksempel modellere brukeren, slik at informasjon som entiteten finner relevant, også vil være relevant for han.



Figur 42: Henlegging av spor – I faste intervaller blir entitens spor sendt til den fysisk-virtuelle oppbevaringsstrukturen, og oppbevart i henhold til lokasjon og tid. Hver dokument i figuren beskriver entitetens kontekstuelle tilstand på et gitt tidspunkt.

Det første en klientapplikasjon gjør når den skal konstruere en ny entitet, er å opprette et kontekstdokument benevnt *baseDocument*. Dette fylles med all relevant informasjon som beskriver hvem eller hva entiteten simulerer. Denne vil danne selve skjelettet for de fremtidige dokumentene entiteten vil publisere om seg selv. Det er først og fremst denne informasjonen andre entiteter vil få frem når de senere aktiverer entitetsdokumentene. Av den grunn bør også postkonteksten i baseDocument beskrive entiteten på en slik måte at andre entiteter kan benytte den til sin egen kontekstspesifikasjon. For eksempel kan postkonteksten i et dokument fra personen Ole inneholde et predikat av typen ”sammen med: ole”. Slike opplysninger kan avsløre sosial kontekst for andre entiteter.

Når dokumentet over er konstruert, leveres det til den nyopprettede entiteten. Dermed får den også et ansikt utad i Condor. Det eneste som rammeverket vil legge til selv ved nye publiseringer er entitetens egen lokasjon og gyldighetstid. Dette vil være en del av dokumentenes aktiveringskontekst. Denne vil også vanligvis endre seg fra publisering til publisering. Mer om gyldighetstiden kommer under.

Neste skritt i entitetkonstruksjonen er å opprette en Context-instanse benevnt *documentPreferences*. DocumentPreferences beskriver filteret for hvilke kontekstdokumenter entiteten selv skal søke etter. Om entiteten simulerer en turist, bør for eksempel personalia, interesser, bekjente, og lignende, være med. I tillegg vil entiteten konstruere sin en egen kontekstinstanse benevnt *baseContext*. Denne vil blant annet fylles med entitetens lokasjon under eksekvering (dette kommer jeg til senere). Sammen vil disse to assosiasjonene definere hva som for entiteten vil bli oppfattet som relevant informasjon.

Alle entiteter har en egen assosiert tidsinstans (fra klassen *Time*). Denne avgjør tiden for når entiteten skal leve. Som et utgangspunkt blir den satt til nøyaktig nåtid. Likevel kan den også settes bakover i tid. Eller fremover. Man kan også bestemme hvor fort tiden skal gå. Det vanlige er at ett sekund for entiteten til-

svarer ett sekund i den fysiske verdenen. Likevel kan denne både settes til å gå raskere, tregere, eller stå helt stille. Man kan også sette den til å gå baklengs. Dette kan muligens fremstå som noe unødvendig funksjonalitet, men hensikten er å tillate konstruksjon av langt mer abstrakte entiteter enn hva en turist for eksempel vil representere. Uansett er entitetens eksistenstid helt avgjørende for hvilke kontekstdokumenter den også vil aktivere. Alle kontekstdokumenter som finnes på serversiden har, som nevnt, assosierte tidsstempler. Disse forteller når de er gyldige (se også Figur 36 og Figur 37). Skrur man tiden bakover, vil man med andre ord også bare aktivere dokumenter som var gyldige rundt dette tidspunktet. Andre, nyere, dokumenter vil for eksempel ikke være det. Analogt anvendes også denne tiden under entitetens egen dokumentproduksjon. Disse vurderes som oppdateringer av hverandre, og får hver for seg et tidsstempel som varer fra tiden dokumentet ble produsert (i henhold til entitetens egen abstrakte tid) og frem til det neste. Slik sett vil hver entitet etterlate seg et kontekstuelt sammenhengende spor i den fysisk-virtuelle oppbevaringsstrukturen (se Figur 42). Om man skrur tiden fremover eller bakover, vil entiteten med andre ord publisere dokumenter som om den ”levde” i den tiden.

Straks baseDocument, baseContext, og entitetstiden er spesifisert, er entiteten klar til å ”leve”. Den vil begynne med dokumentsøk i henhold til sin egen kontekst, og motta relevante dokumenter tilbake. Fra disse vil postkontekst ekstraheres, og bidra til å definere entitetens kontekst ytterligere. Alle mottatte dokumenter vil også midlertidig oppbevares i entitetens *triggeredDocuments*. Her blir de liggende frem til de vurderes som ugyldige. Imidlertid vil et opprinnelig kontekstdokument ligge splittet i flere ulike versjoner. Ett for hvert postkontekstbidrag. Som nevnt vurderes denne konteksten også som entitetens egen, og benyttes derfor under påfølgende dokumentsøk. Bare de bidragene som blir funnet ugyldige, deaktiveres og fjernes.

Alle dokumenter entiteten mottar, videresendes også som kopier ned til klientapplikasjonen (som av den grunn må implementere grensesnittet *EntityListener*). Dette tillater den å utføre dens egentlige funksjonalitet; presentasjon av kontekstrelevant informasjon. Tilsvarende vil den også få beskjed ved deaktiveringer. Det er på et vis dette vi ønsker å oppnå med rammeverket. Muligheten til å presentere kontekstrelevant informasjon i henhold til brukerens kontekst. Entiteten av interesse, er derfor brukeren selv. Det er han denne oppgavens applikasjoner først og fremst må modellere. Likevel er Condors arkitektur relativt generell. Hvilken som helst entitet, abstrakt eller konkret, skal kunne modelleres. Rammeverket er derfor ikke begrenset til å bare omhandle brukeren.

Condors entiteter kan være sammensatte: En entitet kan bestå av flere entiteter. Disse settes sammen ved hjelp av entitetens assosiasjon *consistOf*. Hovedgrunnen til at dette er gjort mulig, er for å tillate entitene å ”leve” i flere tidsepoker simultant. Man kan for eksempel se for seg at hovedentiteten lever i nåtid, mens en underentitet lever et sted i fortid. Den samlede entiteten vil dermed aktivere både historiske dokumenter og nåtidsdokumenter. Et viktig faktum her er imidlertid at postkonteksten som utvinnes hos de forskjellige underentitene (og eventuelt underentitene deres igjen), vil definere også eierentitetens kontekst. For

eksempel vil et dokument aktivert i fortid (av en underentitet) også vurderes som aktivert av eierentiteten. Slik sett kan postkontekst fra en underentitet bidra med å aktivere nye dokumenter også hos eierentiteten. Slike abstrakte entiteter viser seg å være hendige om man ønsker å foreta situasjonssimuleringer. Man kan i slike situasjoner ønske å motta alle sensorobservasjoner (kontekstdokumenter) som blir produsert for nåtid (lokasjon, temperatur, og lignende), men samtidig også aktivere brukerspesifiserte annotering som bare var gyldige i en bestemt fortid.

Ved hjelp av den entiteten beskrevet over, er jeg nå i stand til å konstruere en rekke mer konkrete spesialiseringer som for eksempel sensorer, personer, steder, og lignende. To av disse er allerede lagt til, nemlig *Person* og *Sensor* (se Figur 43). Disse abstraherer vekk hvordan man setter i sammen nye entiteter, ved å tilby enkle metodekall som grensesnitt. Slik sett kan de også vurderes som inkrementer til det opprinnelige kjernerammeverket. Disse tilhører således komponentbiblioteket (jamfør 2.1). For å utvikle nye sensorer, trenger man bare (i de fleste tilfeller) å opprette en instanse av klassen Sensor, og fylle inn dens sensorspesifikke informasjon. Deretter vil det bare være å kalle på *setSensor-Value* for hver nye observasjon som skal publiseres. Resten av dokumentproduk-sjonen overlates til Condor. Om man ønsker at en sensor (eller en annen entitet) kun skal levere observasjoner til oppbevaringsstrukturen uten å søke etter rele-vant informasjon selv, kan dette skrus av ved hjelp av egne funksjonskall hos entiteten. Dette er praktisk å gjøre for statiske sensorer som ikke har behov for å kjenne til sine egne omgivelser (utover det den selv observerer). Det samme er også tilfellet med publiseringen. Denne kan også skrus av om ønskelig. I så fall vil entiteten bli ”usynlig” for andre klientapplikasjoner.

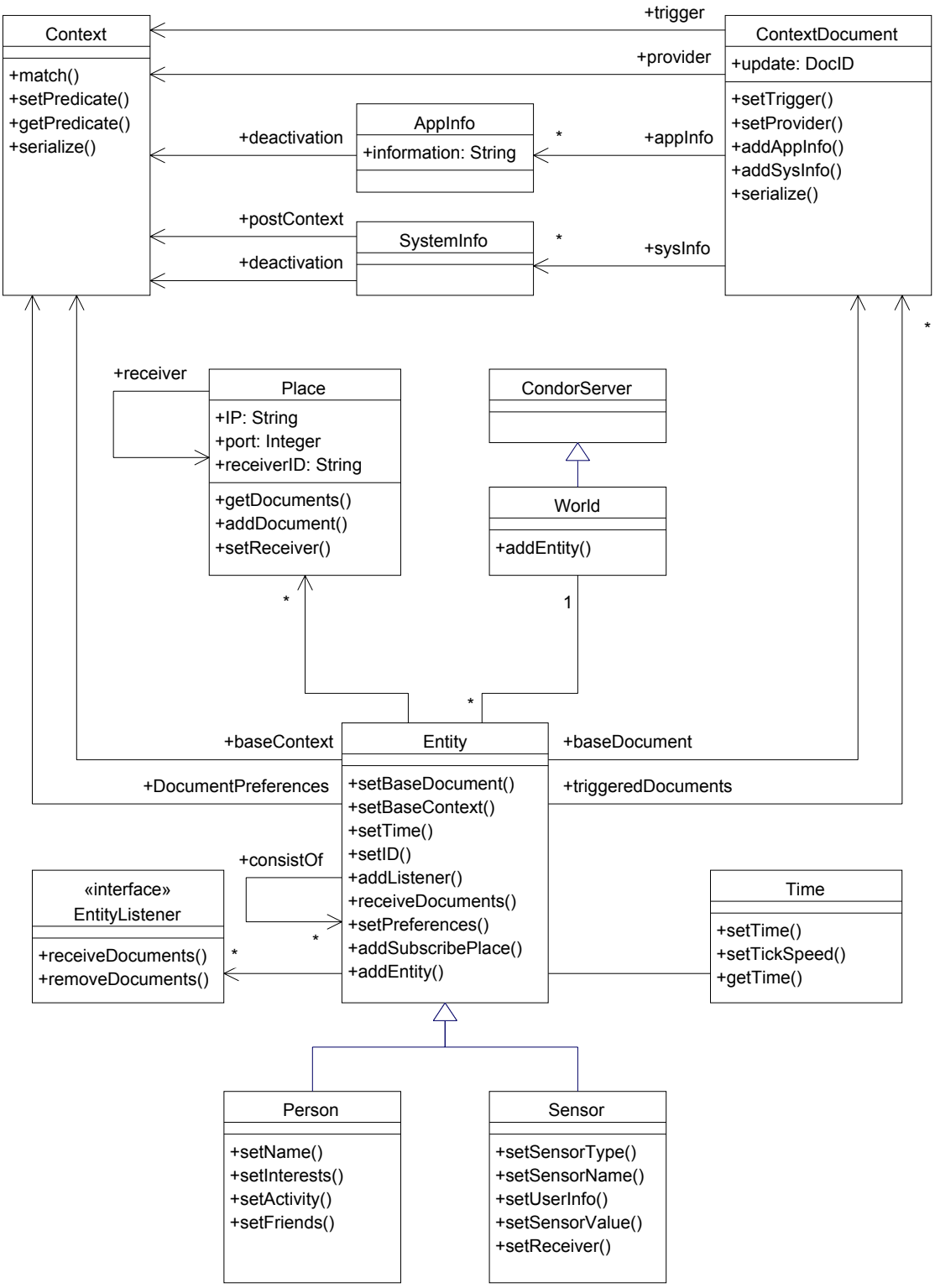
I Kapittel 3.4 drøftet jeg parameteren lokasjon. Denne hadde sin helt spesielle betydning for mobilitetsbaserte applikasjoner. Dette er også tilfellet for den fysisk-virtuelle verdenen i Condor. Ingen kan publisere kontekstdokumenter dit uten at aktiveringskonteksten i dokumentene spesifiserer hvilken konkret fysisk lokasjon de tilhører. Grunnen til dette har med hvordan de oppbevares. I denne ”verdenen” grovsorteres de etter lokasjon. Begrunnelsen for dette kan forsvares om man for eksempel tenker seg Mocado som en klientapplikasjon. Hvis man forsøker å modellere en litt større verden med rundt tusen slike brukere, og alle publiserer et nytt dokument hvert andre sekund (som en følge av GPS-opp-dateringer), blir det fort relativt mye informasjon å holde orden på (rundt 30 000 nye dokumenter for hvert minutt). Dette krever også at man har en viss organi-sering av dokumentene på oppbevaringssiden. Jeg vil gjøre rede for hvordan dette løses i Condor når jeg kommer til beskrivelsen av serversiden.

Kunnskap om fysisk lokasjon angår spesielt de entitetene som simulerer fysiske abstraksjoner. Disse publiserer dokumenter til den fysisk-virtuelle verdenen. Imidlertid oppstår et problem; hvordan kan en entitet avgjøre sin egen lokasjon? Ved vanlig sensoranvendelse, slik som skissert over, vil i utgangspunktet enhver sensorentitet publisere dokumenter til den fysisk-virtuelle verdenen. Dokument-ene forteller selv hvor de kan vurderes som gyldige (i deres aktiveringskontekst). Dette gjør at andre entiteter også kan aktivere dem om de befinner seg på samme

sted (i samme situasjon). Dette systemet lar seg imidlertid vanskelig gjennomføre med lokasjonssensorene. En entitet kan ikke "plukke opp" dokumenter fra en slik sensor i den fysisk-virtuelle verdenen om den ikke kjenner sin egen lokasjon. Analogt er det også bortkastet for en lokasjonssensor å publisere lokasjonsdokumenter til samme sted. Pascoe hevder at *"en lokasjonsverdi i seg selv er meningsløs om den ikke er assosiert til en konkret objekt"* [Pascoe 1998]. I Contextual Information Service (CIS) blir derfor sensorene knyttet direkte opp mot artefaktene[1].

Tilsvarende CIS, har også Condor funksjonalitet for å knytte sensorer direkte opp mot artefakter, her kalt entiteter. Enhver dokumentleverandør kan selv velge hvor den vil sende sine dokumenter. Siden de fleste klientapplikasjonene også har en CondorServer installert, kan de med andre ord omgå serversiden, og isteden sende dokumentene (observasjonene) rett til disse. Man kan med andre ord velge om man vil publisere sine observasjoner via oppbevaringsstrukturen (slik at de blir tilgjengelig for alle), eller direkte, til en eller flere entiteter. For å tillate det siste, må enhver entitet i Condor ha en unik identitet. Denne må settes av klientapplikasjonene selv, og gjøres ved hjelp av metoden *setID* i Entity. Tilsvarende må også dokumentleverandørene kjenne identiteten til adressatene. Dette gjøres ved å konstruere egne Place-instanser som sendes til metoden *addProvidePlace* i Entity. Disse kan som nevnt inneholde både IP og port, men også en mottakeridentitet. I dette tilfellet er kun det siste pålagt. Om en Place-instanse bare har identiteten spesifisert, anvendes IP-multicast (av Place-instansen) for å lokalisere mottakeren. Dette systemet forutsetter at mottakerentitetene befinner seg i "nærheten" på nettverket, noe de vanligvis gjør i slike tilfeller (som for eksempel en GPS-mottaker i forhold til en PDA-bruker).

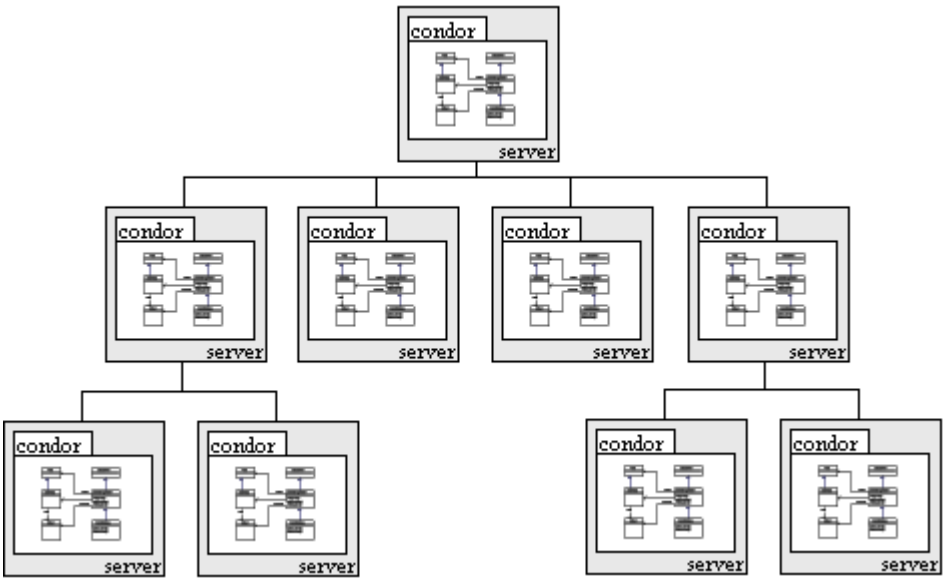
[1] Artefaktene tilsvarer Condors entiteter.



Figur 43: Condors klientarkitektur i sin helhet – Her vises en oversikt over all klassene med tilhørende assosiasjoner på Condors klientside. Person og Sensor tilhører komponentbiblioteket og er derfor inkrementer til rammeverket.

11.3.1 Serversiden

Serversiden av Condor har en noe enklere utforming en klientsiden. Denne delen oppbevarer i hovedsak kontekstdokumenter; funksjonalitet som vanligvis ikke endrer seg fra applikasjon til applikasjon. Serversiden kan også bestå av flere lagringsservere. Disse kan sammen danne hierarkier. Hvert hierarki tilsvarer, som tidligere nevnt, en egen virtuell verden. Dette lar seg best illustrere ved hjelp av et enkelt eksempel. Jeg har allerede drøftet den fysisk-virtuelle verdenen i forhold til entitetene. Denne er en konkret realisering av et slikt hierarki. Hver server i denne vil oppbevare dokumenter innenfor et konkret geografisk område. Hvor stort eller lite dette skal være, avgjøres hos hver server individuelt. Hovedpoenget er at dekningsområde til en subnode alltid skal befinne seg innenfor supernodens område (se Figur 44). En annen måte å se det på, er at hver supernode har ansvaret for ett konkret geografisk område, men delegerer dette ansvaret utover flere underservere. For eksempel kan en supernode som representerer et helt land, si Norge, ha nitten underservere, en for hvert fylke. Disse kan ha underservere hver for seg igjen, som for eksempel én for hver kommune. Disse kan videre inndeles, og slik kan det fortsette til stadig mindre områder er dekket. Én enkelt server på bunnen av hierarkiet (en bladnode) kan derfor ende opp med å oppbevare kontekstdokumenter for et lite og avgrenset geografisk område.



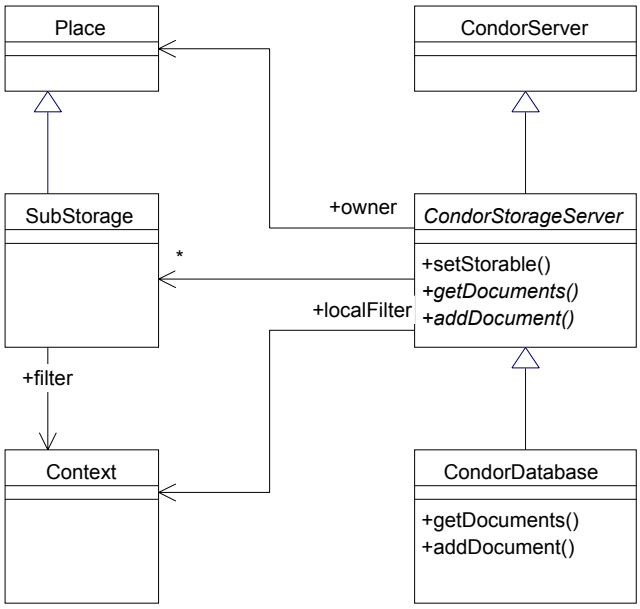
Figur 44: Condors oppbevaringshierarki – Et eksempel på hvordan oppbevaringsstrukturen kan danne et hierarki. Hver node er en lagringsserver med ansvaret for et avgrenset kontekstuelt område. Subnodenes område er alltid innenfor supernodens område. Innholdet i hver server tilsvarer Figur 45.

Den øverste noden i et hierarki er alltid spesiell fordi den representerer summen av hele hierarkiets dekningsområde. Når en klientapplikasjon kontakter oppbevaringsstrukturen for aller første gang (ved hjelp av en Place-instanse), er det

nettopp denne noden den kontakter. Rotserverens adresse må derfor være velkjent blant klientapplikasjonene.

En klientapplikasjon som kontakter serversiden, gjør dette på grunn av et dokumentsøk eller en dokumentpublisering. Dette er de eneste to tjenestene serversiden tilbyr. Forespørselen blir deretter sendt nedover i hierarkiet til riktig oppbevaringsserver identifiseres. Denne vil prosessere forespørselen, og enten sende svardokumenter til returadressen (om det var et søk) eller ta vare på det publiserte dokumentet. Uansett returneres serverens egen adresse til klientapplikasjonen, slik at søkeprosessen kan gjøres noe raskere i de påfølgende transaksjonene (søket kan begynne direkte hos denne noden). Hver lagringsserver har også kjennskap til sin egen supernode, slik at om klienten beveger seg ut av dens dekningsområde, blir forespørslene bare forflyttet ett hakk opp i hierarkiet.

En del av klassene i oppbevaringsstrukturen er allerede beskrevet. Dette gjelder Place, CondorServer, og Context (se Figur 45). Disse er identiske med de som befinner seg på klientsiden. Klassen *CondorStorageServer*, derimot, er spesiell for serversiden, og arver fra CondorServer. Denne omformer dokumentet eller søket til en instanse av henholdsvis ContextDocument eller Context, og avgjør om forespørselen kan behandles der. I motsatt tilfelle sendes den bare videre nedover (eller oppover) i hierarkiet.



Figur 45: Condors oppbevaringsstruktur – Figuren viser alle klassene i oppbevaringsstrukturen sammen med deres assosiasjoner. Denne vil ligge implementert hos hver lagringsserver.

Som tidligere nevnt vil en server ha ansvaret for ett konkret kontekstuelt område. Dette områdets bestemmes lokalt hos hver server, og defineres ved hjelp av kontekstinstansen *localFilter*. Denne beskriver nøyaktig hvilket område serveren er ansvarlig for. Mye av dette kan også delegeres ned til andre eventuelle underservere, og det er heller ikke et krav at disse ikke kan overlappe (med tanke på

ansvarsområde). I slike tilfeller sendes bare en kopi av forespørselen ned til hver enkelt.

CondorStorageServer er i seg selv en abstrakt klasse, og må derfor spesialiseres. Dette blir gjort av klassen *CondorDatabase*. Denne er et inkrement til kjernerammeverket, og har til oppgave å lagre kontekstdokumenter i databaser. Om andre alternative oppbevaringsmetoder ønskes, kan dette realiseres tilsvarende: Ved å spesialisere CondorStorageServer. Dette kan være relevant om man for eksempel ønsker å generere svardokumenter dynamisk. Hvis man i en slik spesialisering ikke ønsker å motta dokumenter for oppbevaring (kun tilby dynamisk genererte dokumenter), kan dette spesifiseres ved hjelp av metoden *setStorable*.

Til sist i serverarkitekturen finner man klassen *SubStorage*. SubStorage er en spesialisering av klassen Place, og peker til en annen underserver i nettverket. SubStorage inneholder en assosiasjon benevnt *filter*. Denne definerer ansvarsområdet for underserveren. Ved hjelp av denne trenger ikke superserveren sende en forespørsel ned til alle sine subserverne, men kun til den eller de som er i stand til å behandle forespørselen.

11.4 Utvikling med Condor

Før jeg drøfter Condor opp mot kravspesifikasjonen, vil jeg se kort på hvordan Mocado og Mobitras kunne ha blitt utviklet dersom rammeverket hadde vært tilgjengelig da jeg gikk i gang med oppgaven. I eksemplene nedenfor vil jeg anta at det eksisterer en implementasjon av den fysisk-virtuelle verdenen som applikasjonene kan benytte, da en slik verden skal foreligge uavhengig av klientapplikasjonene.

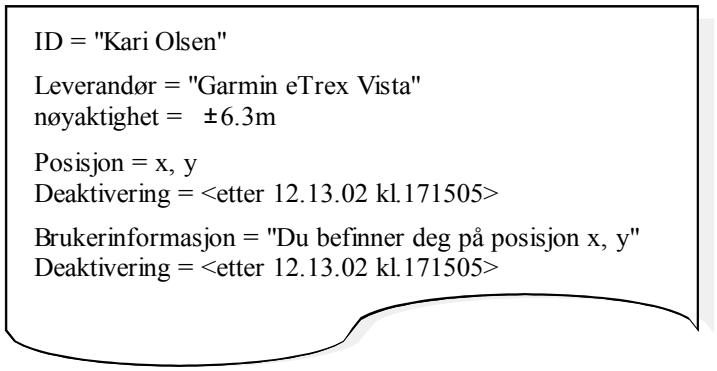
Før jeg går nærmere inn på applikasjonene, vil jeg først se på hvordan en GPS-komponent kunne ha blitt tilgjengeliggjort i rammeverket. En slik komponent vil, som beskrevet ovenfor, kommunisere med en fysisk sensor og produsere dokumenter basert på observasjonene den mottar fra denne. Disse observasjonsdokumentene vil bli sendt over nettverket til en konkret entitet som dokumentet beskriver kontekstuell informasjon for (eller til den fysisk-virtuelle verdenen, dersom observasjonen ble assosiert til en lokasjon). Enhver sensor i Condor vil også realiseres som en egen entitet, og om ønskelig, som en egen applikasjon.

Nedenfor vises et overordnet eksempel på hvordan en GPS-sensor kunne ha blitt implementert som en Condorentitet (eksempelet er beskrevet med utgangspunkt i programmeringsspråket Java):

```
class GPSSensor extends Sensor
implements IPositionSubscriber{
      // Klassen GPSDriver er klassen som kommuniserer med
      // den fysiske GPS-mottakeren, og er ikke en del av
      // Condor. Jeg har lagt ved denne som vedlegg bakerst i
      // oppgaven for ordensskyld. Interfacet IpositionSubscriber
```

```
	// hører også sammen med denne.
	GPSDriver gps;

	public GPSSensor (String mottakerentitet){
		// Denne konstruktøren initsialiserer sensorentiteten, og
		// spesifiserer informasjon som vil bli gitt med
		// publiserte dokumenter som en del av leverandør-
		// informasjonen:
		setSensorType(”Position”);
		setSensorName(”Garmin eTrex Vista”);

		// Setter eniteten ‘mottakerentitet’ til å eie denne
		// sensoren. IP-multicast vil bli benyttet av ramme-
		// verket til å lete opp denne:
		setReceiver(mottakerentitet);

		// Oppretter en instanse av gps-komponenten, og
		// spesifiserer at dette objektet skal motta
		// dens posisjonsoppdateringer:
		gps = new GPSDriver();
		gps.addSubscriber(this);
		gps.startProvide();
	}

	// metoden under kommer fra IpositionSubscriber:
	public void onReceivePosition(UTMPosition pos){
		// Denne metoden mottar alle nye posisjonsoppdateringer
		// fra GPSDriver. Metoden setUserInfo under vil tilsvare
		// brukerinformasjonen i sensordokumentet som publiseres:
		setUserInfo (
			”Du befinner deg på posisjon: ” + pos.toString());

		// Kallet nedenfor resulterer i at et nytt dokument blir
		// produsert og distribuert:
		setSensorValue(pos.toString());
	}
}
```

Komponenten ovenfor er relativt uavhengig av klientapplikasjonene som benytter den. Det eneste som vil variere mellom ulike instansieringer av komponenten, er mottakerentiteten. Denne blir spesifisert som argument til konstruktøren i klassen GPSSensor. Enhver instansiering av komponenten vil dessuten tilsvare en egen fysisk GPS-mottaker tilkoblet maskinen hvor komponenten eksekverer. Sensordokumentene som produseres vil sendes rett til mottakerentiteten, og inneholde informasjonen beskrevet i Figur 46.



Figur 46: En GPS-observasjon – Figuren viser informasjonen som et sensordokument fra GPS-entiteten ville ha inneholdt.

11.4.1 Mocado

I dette avsnittet vil jeg se på hvordan Mocado kunne ha blitt implementert ved hjelp av Condor. Jeg vil ikke gjøre rede for hele applikasjonen, men bare et lite utsnitt som demonstrerer konseptene i rammeverket. Jeg vil anta her at en GPS-entitet, slik beskrevet ovenfor, ligger implementert og produserer dokumenter for entiteten Kari Olsen.

Kodesnutten jeg vil belyse viser hvordan Mocado kunne ha konstruert den tenkte turisten Kari Olsen som en egen entitet[1]. Denne entiteten vil motta posisjons-dokumenter fra GPS-entiteten, og aktivere alle annoteringer (kontekstdokumenter) som Kari (og Karis entitet) kommer i nærheten av. Disse dokumentene vil bli sendt ned til metoden receiveDocuments i klassen Kari. Jeg vil også vise hvordan en annotering kan konstrueres som et kontekstdokument fra klientapplikasjonen.

```
Class Kari extends Person
Implements EntityListener
{
      public Kari(){
            // Metoden konstruerer en entitet som skal representere
            // Kari, og spesifiserer hennes navn (som også blir hennes
            // ID), og aktivitet. Kun dokumenter som har noe med
            // turisme å gjøre skal aktiveres:
            setName("Kari Olsen");
            setActivity({"turist"});

            // Spesifiserer at dette objektet skal motta alle
            // dokumenter som entiteten aktiverer:
            addListener(this);

            // Gjør entiteten "levende" ved å registrere den
            // i World. Da vil den være i stand til å motta
            // aktiverte dokumenter fra oppbevaringsstrukturen,
            // og observasjonsdokumenter fra GPS-entiteten:
            Condor.World.addEntity(this);
      }
```

[1] Jeg har gjort dette eksempelet meget konkret (fremfor generelt) slik at det blir lite og konsist.

```
	public void receiveDocuments(ContextDocument[] dokumenter){
		// Denne metoden vil bli kalt fra rammeverket
		// hver gang turisten aktiverer nye dokumenter, og
		// skal resultere i dokumentenes informasjon blir
		// fremvist.
	}

	public void removeDocuments(ContextDocument[] dokumenter){
		// Denne metoden vil bli kalt fra rammeverket
		// hver gang tidligere aktiverte dokumenter skal
		// deaktiveres.
	}

	public void lagAnnotering(Sting informasjon, UTMPosition pos){
		// Denne metoden mottar en tekst som Kari har
		// forfattet, og lager en annotering av den.
		// Denne annoteringen får ingen postkontekst siden
		// det er uvisst hva informasjonen beskriver.
		ContextDocument doc = new ContextDocument();

		// Her spesifiserer jeg den kontekstuelle situasjonen
		// som må inntreffe dersom annoteringen skal aktiveres:
		Context trigger = new Context();
		Trigger.setPredicate("location = " + pos.toString());
		Trigger.setPredicate("within = 10m");
		Trigger.setPredicate("from = 18.05.02 kl.1324");
		doc.setTrigger(trigger);

		// Her spesifiserer jeg hvem som har produsert dokumentet:
		Context provider = new Context();
		provider.setPredicate("Forfatter = Kari Olsen")
		doc.setProvider(provider);

		// Her spesifiserer jeg brukerinformasjonen som skal
		// assosieres med annoteringen:
		SysInfo userInformation = new SysInfo();
		deactivation = new Context();
		deactivation.setPredicate("location = " + pos.toString());
		deactivation.setPredicate("not within = 15m");
		userInformation.deactivation = deactivation;
		// Gir med metodens argument som innhold:
		userInformation.information = informasjon;
		doc.addSysInfo(userInformation);

		// Sender dokumentet av gårde til den fysisk-virtuelle
		// verdenen:
		Place sted = new Place();
		Sted.setReceiver(<adressen til øverste node i
				oppbevaringshierarkiet>);
		sted.addDocument(doc);
	}
}
```

Klassen Kari ville ha erstattet mye av arkitekturen presentert i Figur 49 Figur 51 (se Vedlegg). Eksempelet demonstrerer derfor at en vesentlig del av kildekoden i Mocado kunne ha vært utelatt om Condor hadde vært tilgjengelig under utviklingen.

11.4.2 Mobitras

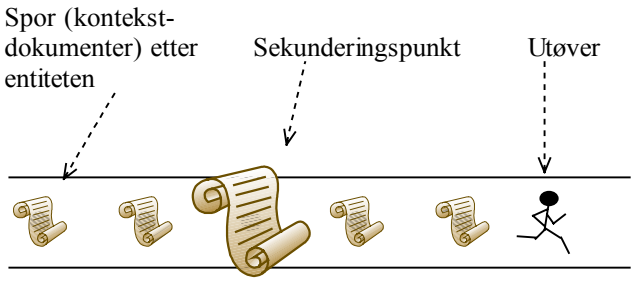
Likt som for Mocado vil jeg også belyse hvordan Mobitras kunne ha vært utviklet ved hjelp av Condor. En av grunnene for at velger å beskrive begge applikasjonene er for å vise at rammeverket er relativt generelt og kan passe for flere ulike applikasjonstyper.

Kildekoden nedenfor viser hvordan den nye versjonen av Mobitras klargjør utøveren Kari for en ny trening. Når en trening senere igangsettes, vil hun begynne å aktivere kontekstdokumentene langsetter løypen etter hvert som hun løper. Dette inkluderer først og fremt løypens sekunderingspunkter, men også skyggeannoteringer fra de valgte skyggene. I eksempelet vil jeg ikke implementere treningshendelser. Dette har jeg valgt å unnlate ene og alene for å minske kompleksiteten til eksempelet.

```
Class Kari extends Person
{
      public Kari(){
            // Lager opp entiteten Kari Olsen:
            setName(”Kari Olsen”);
            setActivity({”trening”});
            leggTilSkygge(“Vebjørn Jensen”, ”06.03.02 kl.1904”);
            leggTilSkygge(“Kari Olsen”, ”11.05.02 kl.1321”);
      }

      public void leggTilSkygge(skygge, tid){
            // Denne metoden skaper en ny entitet, og lar denne være
            // en del av Kari. Den nye entiteten skal settes til å
            // være interessert i dokumenter produsert av skyggen.
            // Setter også entiteten til å eksistere på nøyaktig
            // samme tidspunkt som opptaket da opptaket ble foretatt.
            // På den måten vil Kari aktivere alle dokumentene skyggen
            // produserte mens den løp:
            Person p = new Person;
            p.setName(this.getName());
            p.setFriends(skygge);
            p.setActivity(“trening”);

            // Definerer nøyaktig tidspunktet for når
            // skyggen startet løpet sitt, men
            // stopper tiden inntil løpet begynner:
            Time t = new Time();
            t.setTime(tid);
            t.setTickSpeed(0);
            p.setTime(t);

            // Legger til skyggen som en del av utøveren:
            addEntity(p);
      }

      public void receiveDocuments(ContextDocument[] dokumenter){
            // Denne metoden vil kalles fra rammeverket hver gang
            // utøveren aktiverer nye sekunderingspunkter eller
            // løypehendelser. Dette betyr blant annet at
            // informasjonen som er aktivert skal leses opp.
      }
```

```
	public void removeDocuments(ContextDocument[] dokumenter){
		// Denne metoden vil bli kalt fra rammeverket
		// hver gang tidligere aktiverte dokumenter skal
		// deaktiveres. I denne applikasjonen ville ikke denne
		// være hensiktsmessig fordi informasjon pre-
		// senteres umiddelbart etter aktivering, og bare da.
	}
}
```

Underveis under en trening vil Kari henlegge nye kontekstdokumenter langsetter løypen. Dette vil inntreffe etter hvert som hennes entitet mottar posisjonsoppdateringer fra GPS-entiteten (se Figur 47). Dermed vil også løypen fylles med dokumenter som sammen lager et kontekstuelt spor etter Kari. Det er nettopp dette sporet andre utøvere også vil aktivere dersom de senere tar hennes skygge med under en av sine egne treninger.



Figur 47: Spor i Mobitras – Figuren viser hvordan utøveren vil henlegge et spor i form av kontekstdokumenter under en trening i Mobitras. Langs ruten vil også sekunderingspunkter ligge plassert.

Den nye klassen presentert ovenfor ville ha eliminert behovet for en god del av den eksisterende arkitekturen til Mobitras. Dette inkluderer først og fremst arkitekturene som kan ses i figur 61 og 62, men også mye av arkitekturen fra figur 57, 58, 59, og 60 (se Vedlegg). Condor ville derfor ha redusert utviklingstiden for Mobitras vesentlig om det hadde vært tilgjengelig da jeg gikk i gang med utformingen av applikasjonen.

11.5 Drøfting av Condor opp mot kravspesifikasjonen

Jeg vil avslutte dette kapitlet med å drøfte Condor opp mot kravspesifikasjonen. Selv om Condor fra begynnelsen av ble konstruert med disse kravene som utgangspunkt, ble ikke nødvendigvis alle innfridd. Jeg vil uansett drøfte alle kravene opp mot rammeverket nedenfor.

Det første kravet i spesifikasjonen gjaldt anvendelse av eksterne sensorsystemer. Dette var systemer som registrerte kontekst ved hjelp av eksternt tilkoblet utstyr.

Blant oppgavens applikasjoner var dette nesten ensbetydende med GPS-mottakere. Inspirert av Context Toolkit (jamfør 10.3) har jeg også forsøkt å unngå en tett assosiasjon mellom sensorer og applikasjoner. Ved å kun tillate kommunikasjon ved hjelp av kontekstdokumenter, mener jeg også å ha lykkes med det. Ved hjelp av dette systemet kan man legge til både interne og eksterne sensorløsninger uten å vite hvilke applikasjoner som kan komme til å anvende deres observasjoner. Eneste unntak er lokasjon. Sensorene som tilbyr dette, må vite hvilke entiteter de tilbyr lokasjon for. Uansett kan alle sensorløsningene ligge distribuert på egne maskiner og eksekvere i egne prosesser. Til sammen tillater, og forenkler, dette gjenbruk av både interne og eksterne sensorløsninger. Krav 1 og 2 vurderer jeg derfor som tilfredsstilt.

I Condor blir sensorsystemer vurdert som entiteter. De kan både være kontekstoppmerksomme i forhold til sin egen situasjon, og samtidig registrere kontekstuell informasjon for andre. Dette gjør at de også kan registrere om sensoren de representerer for eksempel endrer lokasjon. Dette har jeg beskrevet som relevant tidligere, med et termometer som hang fast på en bil: Denne ville endre dekningsområde i takt med bilens lokasjon (jamfør 7.1.1). Dermed får et sensorsystem ikke bare muligheten til å feste en aktiveringskontekst til sine observasjoner, men også å endre på denne i henhold til sin egen kontekstuelle situasjon. En entitet som ønsker å benytte disse observasjonene, vil med andre ord være avhengig av at aktiveringskonteksten hos disse overensstemmer med dens egen kontekstuelle situasjon. Kun da vil de bli anvendt. Derfor innfris også krav 3, betinget anvendelse av sensorsystemer.

En betydelig forskjell som skiller Condor fra de andre rammeverkene jeg har studert, er hvordan tjenestekvalitet behandles. Fremfor at dette i Condor legges på sensornivå, legges det isteden på observasjonsnivå. For eksempel vil rammeverket aldri vurdere å anvende et bestemt sett med sensorer konsekvent. Isteden vil den anvende de observasjonene som til enhver tid vurderes som best (krav 4). Dette vil gjøres i henhold til klientutviklernes egne prioriteringer, fastsatt i entitetene. Dermed vil man også unngå problemet med dekning; kontekstleverandører som ”lover” mer enn de kan holde [Dey 2000]. De får aldri muligheten til å gjøre dette. Condor vurderer ikke leverandørene, bare deres distribuerte kontekstdokumenter. Inne i hver av disse vil både leverandørinformasjon og tjenestekvalitet være fastsatt. Om det siste endres fra observasjon til observasjon (noe som er vanlig for GPS-mottakere) vil dette også avspeiles her. Krav 4, dynamisk skifte av sensorsystemer, blir derfor innfridd.

En av Condors fremste egenskaper er behandling av entiteter. Enhver entitet kan fortelle om seg selv til andre, ved hjelp av å publisere kontekstdokumenter. Disse oppbevares i den fysisk-virtuelle verdenen, slik at entitetene vil aktivere hverandres kontekstdokumenter under eksekvering. I tillegg skiller ikke Condor mellom sensorer, personer, gjenstander, eller andre entiteter. Et dokument som beskriver en person, behandles likt som en observasjon fra et sensorsystem. Det vil si, rammeverket uthenter (blant annet) kontekstuell informasjon fra begge. Et dokument fra en personentitet kan for eksempel innehold opplysninger om personens personalia, posisjon, hjerterytme, interesser, og lignende. Dermed vil

også utveksling av sosial kontekst mellom entitetene i Condor oppfylles naturlig (krav 5).

Når det gjelder krav 6, privat anvendelse av kontekstdokumenter, er dette i Condor ufullstendig adressert. En metode som imidlertid kan anvendes, er å opprette en egen dedikert oppbevaringsserver for akkurat den entiteten det gjelder (eller alternativt skru av dens publisering). Denne serveren vil i så fall ingen andre entiteter ha adgang til. Uansett er dette ingen god løsning. Helst bør man kunne detaljstyre hvem som skal ha tilgang til de forskjellige publikasjonene, og når [Chen et al. 2000]. I tillegg bør systemet hindre klienter fra å kunne utgi seg for å være noen andre (eller annet) enn den de er. Et slikt system er ikke blitt skissert. Jeg vil av den grunn ikke vurdere dette kravet som innfridd.

Krav 7 og 8 var behovet for henholdsvis manuell kontekstspesifikasjon og overstyring. Dette var spesielt relevant i tilfeller der entiteten var brukeren selv (som turisten i Mocado). Begge disse kan innfris ved hjelp av en entitets baseContext og time instanser. Her kan man fastsette nøyaktig hvilken basissituasjon man ønsker at entiteten skal befinne seg i. Man har også mulighet til å overstyre dens posisjonering, eller andre kontekstverdier, og prioritere anvendelse av observasjoner fra gitte sensorsystemer. Begge kravene blir derfor tilfredsstilt.

Tidligere registrert kontekst kan være med på å utdype en entitets situasjon i nåtid. Likevel vil ikke en entitet ivareta alt av postkontekst som den en gang har funnet relevant; dette ville resultert i store mengder med informasjon sett i forhold til PDA-enes lagringskapasitet. Isteden må kontekstdokumentene selv indikere om informasjonen de kommer med er verdt å lagre, og i så fall for hvor lenge. Dette er noe av hensikten med deres postkontekst. En observasjon fra en GPS-sensor kan for eksempel ha to postkontekstbidrag: Ett som forteller om mottakerentitetens posisjon, og som deaktiveres når en ny observasjon blir gjort tilgjengelig, og ett som forteller at entiteten på ett eller annet tidspunkt har befunnet seg på den aktuelle posisjonen tidligere, og som deaktiveres etter et år. Det siste bidraget vil beskrive entitetens situasjon ett år frem i tid, og fungere som en del av dens historiske kontekst. Om et annet kontekstdokument på et senere tidspunkt spesifiserer at entiteten (som forsøker å aktivere dokumentet) skal ha befunnet seg på dette stedet tidligere, vil dette bidraget kunne gi et svar på om dette kravet blir tilfredsstilt. Krav 9 blir derfor imøtekommet.

Condor har strengt tatt ingen særegen funksjonalitet for å tilby kontekstutledning (krav 10), men dette inntreffer likevel naturlig slik entitetene fungerer. De søker etter kontekstdokumenter i henhold til sin egen kontekst, og trekker ut kontekstuell informasjon fra svardokumentene som returneres. Dette vurderes som entitetens egen kontekst, og blir derfor en del av søkefilteret i entitetens påfølgende dokumentsøk. Dette vil pågå i en evig prosess. Dermed får man også en slags kontinuerlig repeterende kontekstutledning. Likevel kan ikke all kontekstutledning løses ved hjelp av statiske kontekstdokumenter. Om man for eksempel ønsker å omforme en temperaturverdi fra celsius til fahrenheit, blir dette vanskelig. Begge disse parametrene tilhører kontinuerlige skalaer (jamfør 3.3). Dermed kan ikke en slik omforming beskrives med kontekstdokumenter alene. Dette kan isteden løses ved hjelp av å konstruere egne spesialiseringer av klassen Dynamic-

Storage på serversiden. Disse er spesielt tiltenkt slik bruk. Her kan komponenter, i likhet med Interpreters i Context Toolkit, konstrueres for å ta seg av omformingen (jamfør 10.3). Når disse mottar dokumentforespørslene, kan de generere svardokumentene dynamisk. Oppsummert vil derfor hele krav 10 blir innfridd.

Det siste kravet for behandling av kontekst, krav 11, dreide seg om abonnering og spørring. Det siste, spørringen, kan i Condor gjøres både direkte ved hjelp av et Place-objekt, eller indirekte ved hjelp av en entitet. Det første, abonneringen, er imidlertid gjort litt utilgjengelig. Det lar seg faktisk ikke gjøre å abonnere etter kontekstdokumenter direkte fra en leverandør. Dette har med filosofien bak Condor å gjøre; en klientapplikasjon skal ikke ha behov for et bundet abonneringsforhold til en leverandør. Valg av informasjon skal heller ligge på dokumentnivå. Det er dokumentene som er interessante, ikke leverandørene. Disse kan komme og gå i henhold til entitetens kontekst. Derimot kan en leverandør velge å levere dokumenter direkte til en entitet. Dette forholdet vil for entiteten (mottakeren) være ukjent. Alle dokumenter den mottar kommer samme vei uansett. Effekten vil likevel bli den samme som ved abonnering. Jeg vurdere derfor krav 11 som imøtekommet.

Condor er spesielt utviklet for å kunne anvende kontekstrelevant informasjon. Dette var nedfelt som et eget krav i kravspesifikasjonen (krav 12). Videre var det også fastsatt at det skulle foreligge et system for å kunne uthente kontekstuelle opplysninger fra informasjonen (krav 13). I Condor er dette realisert direkte. Kontekstdokumentene har et eget separat felt hvor informasjonens postkontekst kan bli spesifisert. Dermed har både klientsiden av Condor og klientapplikasjonene mulighet for å uthente kontekstuelle opplysninger fra aktivert informasjon. Både krav 12 og 13 blir derfor besvart tilfredsstillende.

Bakgrunnen for krav 14, muligheten for spesifikasjon av dynamisk informasjon, kom som en følge av at enkelte annoteringer i Mocado ikke kunne beskrives som statisk informasjon. I Kapittel 8.1.3 drøftet jeg blant annet hvordan enkelte annoteringer kunne skifte geografisk lokasjon over tid. I Condor løses dette indirekte. Fremfor å vurdere et kontekstdokument som dynamisk, ses dens endringer heller på som et utgangspunkt for et nytt dokument. Hvis en annotering for eksempel skifter lokasjon, så endres ikke det gamle dokumentet (på serversiden). Det produseres heller et nytt, mens det gamle isteden lukkes tidsmessig.

Anvendelse av konteksthistorikk kan vurderes som to krav: Informasjonshistorikk og aktiveringshistorikk (krav 15). Det første realiseres ved at all informasjon som produseres blir oppbevart som dokumenter i en oppbevaringsstruktur. Dette er en av Condors hovedegenskaper. Når det gjelder aktiveringshistorikk, så foreligger ikke noe eget system for la en entitet ivareta en logg over aktiverte dokumenter. Likevel har man mulighet til å modellere dette gjennom kontekstdokumentene. Tanken er at dokumentforfatterne selv best vet hvordan reaktiveringer av et dokument skal behandles. Spørsmålene de må kunne besvare er av typen: ”Skal det samme dokumentet presenteres ved andre og tredje reaktivering?”. Om svaret er nei: ”Hvilke andre dokumenter skal i så fall presenteres?”. Disse svarene kan ikke avgjøres av Condor. Dette vil variere fra applikasjon til applikasjon. Isteden bør svarene innlemmes i dokumentene selv. Dette kan gjøres ved å

manipulere aktiveringskonteksten og postkonteksten hos dokumentene det gjelder. Jeg vil illustrere dette med et eksempel. Eksempelet viser hvordan man kan forfatte to ulike kontekstdokumenter som skal presenteres ved henholdsvis første og andre aktivering. Det første dokumentet under vil bli presentert ved første aktivering (anta her at man ankommer Tusenfryd, og variabelen ”aktivering” er lik null):

<table>
<tr><td>Aktiveringskontekst</td><td>Sted = <innenfor Tusenfryds areal>, aktivering = 0</td></tr>
<tr><td>Postkontekst</td><td>
<table>
<tr><td>Kontekstbidrag</td><td>aktivering = aktivering + 1</td></tr>
<tr><td>Deaktiveringskontekst</td><td><Ett år etter aktivering></td></tr>
</table>
<table>
<tr><td>Kontekstbidrag</td><td>Sted = ”Tusenfryd”</td></tr>
<tr><td>Deaktiveringskontekst[1]</td><td>Sted != Tusenfryd</td></tr>
</table>
</td></tr>
<tr><td>Brukerinformasjon</td><td>
<table>
<tr><td>Informasjon</td><td>”Norgesparken Tusenfryd, fornøyelsespark i Ås kommune, Akershus, ved E6 og E18 nord for Vinterbro”</td></tr>
<tr><td>Deaktiveringskontekst</td><td>Sted != Tusenfryd</td></tr>
</table>
</td></tr>
</table>

Etter at dokumentet over er blitt aktivert, vil konteksten til entiteten som aktiverte dokumentet, suppleres med opplysningene ’Sted = ”Tusenfryd”’, og ’aktivering = 1’. Det siste gjør at dette dokumentet ikke vil aktiveres neste gang brukeren ankommer tusenfryd, dersom dette skjer innen et år (siden aktiveringskonteksten til dokumentet spesifiserer at denne variablen må være lik null). Isteden vil det neste dokumentet bli aktivert:

[1] Tegnet ’!=’ leses ”ikke lik”.

<table>
<tr><td>Aktiveringskontekst</td><td><innenfor Tusenfryds areal>, aktivering > 0</td></tr>
<tr><td>Postkontekst</td><td>
<table>
<tr><td>Kontekstbidrag</td><td>aktivering = aktivering + 1</td></tr>
<tr><td>Deaktiveringskontekst</td><td><Ett år etter aktivering></td></tr>
</table>
<table>
<tr><td>Kontekstbidrag</td><td>Sted = ”Tusenfryd”</td></tr>
<tr><td>Deaktiveringskontekst</td><td>Sted != Tusenfryd</td></tr>
</table>
</td></tr>
<tr><td>Brukerinformasjon</td><td>
<table>
<tr><td>Informasjon</td><td>”Du befinner deg i parken Tusenfryd”</td></tr>
<tr><td>Deaktiveringskontekst</td><td>Sted != Tusenfryd</td></tr>
</table>
</td></tr>
</table>

Oppsummert vil systemet presentert ovenfor gjøre at krav 15 blir innfridd.

Når det gjelder krav 16, separate deaktiveringskontekster, så adresseres dette i kontekstdokumentene direkte. All brukerinformasjon og postkontekst kan inndeles i flere seksjoner i samme kontekstdokument, hvorav hver og en også kan ha sin egen separate deaktiveringskontekst.

Det å kunne publisere nye kontekstdokumenter under eksekvering, tillates i Condor naturlig (krav 17). Dette er noe av hensikten med oppbevaringsstrukturen. Denne er sammensatt av selvstendige servere som eksekverer uavhengig av klientene. De mottar og returnerer dokumenter kontinuerlig. Slik sett blir et hvert nytt dokument som en leverandør leverer, umiddelbart gjort tilgjengelig for alle klientapplikasjoner (krav 18). Når det gjelder krav 19, privat dokumenttilgjengelighet, blir dette analogt med krav 6, privat kontekstanvendelse. Siden Condor sammenfatter behandlingen av kontekstrelevant informasjon med behandlingen av kontekst, vil disse områdene dele mange av de samme fordelene og ulempene. Krav 19 er et typisk eksempel på dette. Ettersom rammeverket ikke adresserer noe godt system for å behandle privat kontekst, vil dette også gjelde privat brukerinformasjon. Av den grunn blir heller ikke krav 19 innfridd.

Det siste kravet, krav 20, dreide seg om brukerens uavhengige tilgjengelighet til kontekstrelevant informasjon (i forhold til hans kontekstuelle situasjon). Condor er hovedsakelig tilrettelagt for å skape og anvende entiteter som hver for seg henter frem (og tilbyr) kontekstrelevant informasjon. Likevel kan klientapplikasjonene gjerne benytte rammeverket uten entitetene. Enhver klientapplikasjon kan opprette et eget kontekstdokument og sende dette direkte til en oppbevaringsstruktur (dette demonstrerte i Mocado i Kapittel 11.4.1). Analogt kan man også søke etter dokumenter tilsvarende (ved å opprette en søkekontekst og sende det direkte til en oppbevaringsstruktur). I tillegg kan entitetene manipuleres ved situasjonssimulering. Man kan i slike situasjoner simulere at man for eksempel befinner seg på et annet sted enn det som er tilfellet, og dermed også få tilgang til informasjon uavhengig av sin egentlige kontekst. Krav 20 blir derfor tilfredsstilt.

Krav som tilfredsstilles:	1-5, 7-18, 20
Krav som ikke tilfredsstilles:	6, 19

11.6 Oppsummering

I dette kapitlet har jeg presentert og drøftet Condor som et rammeverk. Jeg begynte således med en viktig observasjon; det viste seg at behandlingen av kontekst delte mye av de samme egenskapene som behandlingen av kontekstrelevant informasjon (og omvendt). Begge kunne vurderes som brukerinformasjon, og begge kunne vurderes som kontekst. Hvordan man betraktet det, var isteden avhengig av hva man skulle anvende det til. Dette kunne i tillegg variere fra situasjon til situasjon. Dette førte meg videre til en idé om å integrerer disse områdene tettere. Fremfor å skille de ut i to separate lag, ble de isteden integrert sammen til ett. Sentralt i denne løsningen sto kontekstdokumentene.

Fra idé gikk jeg videre til konstruksjon. Denne ble delt inn i to deler; klientsiden og oppbevaringsstrukturen. Det siste var et nettverk av lagringsservere som kunna motta, oppbevare, og returnere kontekstdokumenter i henhold til kontekst. Disse kunne settes sammen til å danne ulike hierarkier, og således utgjøre forskjellige virtuelle verdener. En spesiell realisering her var den fysisk-virtuelle. Denne kunne tenkes på som en modellering av den fysiske verdenen med entiter som innbyggere, og skulle være felles for alle klientapplikasjoner og entiteter utviklet med rammeverket. Strukturen dannet videre et hierarki som delte den fysiske verdenen inn i lokasjonsbestemte soner.

En klient i Condor kunne bli vurdert som enten en dokumentleverandør, en dokumentforbruker, eller begge deler. I så tilfelle kunne den realiseres som en entitet. Personer og sensorer var typiske eksempler på entiteter. Disse produserte dokumenter om seg selv til den fysisk-virtuelle verdenen, og mottok dokumenter tilbake i henhold til sin egen kontekst. Disse inneholdt både brukerinformasjon, som skulle ned til klientapplikasjonene, og postkontekst for videre konkretisering av entitetenes egen situasjon.

Til sist drøftet jeg Condor opp mot kravspesifikasjonen. De fleste punktene i denne ble adressert tilfredsstillende, med unntak av de som dreide seg om privat kontra offentlig oppbevaring av informasjon.

Kapittel 12

Konklusjon og videre arbeid

Jeg begynte første Kapittel med en redegjørelse for hvilke typer applikasjoner oppgaven skulle dreie seg om. Dette var mobilitetsbaserte applikasjoner som var kontekstoppmerksomme for å anvende kontekstrelevant informasjon. Jeg presenterte noen fremtredende egenskaper hos disse, og drøftet kort behovet utviklingen av slike har for et rammeverk. Deretter ble oppgavens problemstilling presentert. Den lød:

> *Jeg vil studere mobilitetsbaserte kontekstoppmerksomme applikasjoner som behandler kontekstrelevant informasjon, og hvordan et rammeverk for slike kan utformes.*

12.1 Konklusjon

Etter å studert oppgavens område har jeg kommet frem til at kontekstuell-, og kontekstrelevant informasjon bør vurderes som samme abstraksjon. Avhengig av hvordan dette anvendes, inneholder begge typene som regel både kontekstuelle og kontekstrelevante opplysninger. Dette gjør at de også deler mange av de samme egenskapene. Blant annet er begge avhengige av en spesifikasjon som forteller når informasjonen kan vurderes som gyldig, og når den etter dette vil bli ugyldig igjen. På grunn av disse forholdene vil jeg konkludere med at anvendelse av kontekstoppmerksomhet og kontekstrelevant informasjon bør integreres under en felles abstraksjon, i et felles rammeverk. Dette konseptet demonstrerte jeg i løsningsforslaget Condor ved hjelp av det jeg benevnte som kontekstdokumenter. Ved å oppbevare diskontinuerlige mengder med kontekstuell informasjon på denne måten, oppnådde man i hovedsak to egenskaper: Man fikk en enkel informasjonsutveksling som tok hensyn til tjenestekvalitet, aktiveringskontekst, og

deaktiveringskontekst, og man fikk en klar separasjon mellom de ulike kontekstleverandørene og abonnentene i systemet.

Kravspesifikasjonen i Kapittel 9 avslørte et vesentlig behov for oppbevaring og utveksling av kontekst og kontekstrelevant informasjon mellom ulike klientapplikasjoner. Dette tilgjengeliggjorde blant annet anvendelse av historisk og sosial kontekst, i tillegg til situasjonssimulering. Dette var alle sentrale krav som gikk igjen hos eksempelapplikasjonene. Dette ble løst i Condor ved hjelp av to konsepter; oppbevaring av kontekstdokumenter i en eller flere atskilte og distribuerte oppbevaringsstrukturer, og muligheten for å la klientapplikasjonene skape/simulere ”levende” entiteter med tilhørende abstrakte tidsbegreper. Ved det første konseptet, oppbevaringsstrukturene, drøftet jeg også spesielt en realisering av denne benevnt som den ”fysisk-virtuelle” verdenen. Denne var ment som å danne et felles kommunikasjonsmedium for alle personer, sensorer, dyr, gjenstander, steder, eller andre både fysiske og abstrakte entiteter som klientapplikasjonene kunne ha behov for å simulere og motta relevant informasjon for. Ved å strukturere denne hierarkisk, i henhold til lokasjon, kunne man også sannsynligvis oppnå bedre effektivitet med tanke på søketid. Til sist demonstrerte Condor hvordan entitetene kunne spesialiseres til inkrementer som Person og Sensor. Dette medførte at løsningen ble mer i retning black box.

Idéene ovenfor utgjorde ryggraden i Condor. Sammen med utfyllende komponenter dannet disse i felleskap et overordnet forslag på et helhetlig rammeverk. Om dette rammeverket hadde vært tilgjengelig under utviklingen av Mocado og Mobitras, fant jeg at utviklingstiden og arkitekturkompleksiteten til disse ville ha blitt vesentlig redusert. Det samme indikerte analysen av Condor opp mot kravspesifikasjonen. Jeg vil derfor konkludere med at Condor kan støtte utviklingen av mobilitetsbaserte kontekstoppmerksomme applikasjoner som anvender kontekstrelevant informasjon, med det meste av funksjonaliteten jeg for slike applikasjoner har funnet relevant (18 av 20 krav).

12.2 Videre arbeid

Under presentasjonen av Condor drøftet jeg to krav jeg vurderte som ikke tilfredsstilt (jamfør 11.5). Dette var krav 6 og 19. Disse vil imidlertid smeltes sammen til ett og samme krav når kontekst og kontekstrelevant informasjon flettes sammen under en felles abstraksjon. Krav 6 lød: ”Om kommunikasjon av kontekst er mulig, bør man også ha mulighet for å helt, eller delvis, begrense denne distribusjonen under eksekvering. Mye av en brukers kontekst kan til ulike tider, og ovenfor ulike personer, være av sensitiv art, slik at dette ikke ønskes utvekslet fritt.”. Slik Condor er fremstilt i denne oppgaven, er det ikke beskrevet noe funksjonalitet som kan ivareta dette. Hvordan kan man bestemme hvilke entiteter som til hvilke tider skal ha tilgang til publiserte kontekstdokumenter? Hvordan kan man være sikker på at en entitet faktisk er den som den utgir seg for å være? Jeg vurderer dette som et reelt problemområde for videre arbeid. Likevel

er det først og fremst det faktum at Condor i løpet av oppgaven aldri blitt implementert, som bør fremheves. En kravspesifikasjon er fremsatt, andre eksisterende rammeverk er blitt studert, og et løsningsforslag er skissert. Det mest naturlige skrittet videre er derfor å foreta en konkret realisering av arkitekturen [Fayad et al. 1999] (jamfør 2.2). Dette innebærer blant annet at man er nødt til å avgjøre hvordan klassene og metodene i Condor skal implementeres. Spesielt sentralt her er aktiveringsprosessen. Hva slags algoritme bør metoden match i klassen Context følge? Det er her avgjørelsen om hvorvidt en kontekstspesifikasjon er gyldig i forhold til en annen besluttes. Jeg har heller ikke skissert noe forslag på hvordan formatet på konteksten i kontekstdokumentene skal se ut. Her bør man følge en felles standard. Hvordan kan for eksempel en entitets lokasjon representeres ved hjelp av en kontekstspesifikasjon? Det er viktig at alle klientapplikasjonene og oppbevaringsstrukturene er implementert slik at de er ”enige” om dette formatet. Dokumentene er, som nevnt, den eneste måten distribuerte komponenter kan kommunisere på i Condor. Uten en felles standard på disse vil mye av støtten som rammeverket tilbyr falle bort.

Som nevnt i begynnelsen av oppgaven vil et rammeverk stadig fortsette å utvikle seg etter en initsiell implementasjon [Fayad et al. 1999] (jamfør 2.2). Hva som mangler, eller ikke er blitt løst tilfredsstilende, kan vanskelig avgjøres uten å utvikle noen konkrete klientapplikasjoner. Derfor vil denne prosessen også pågå i mange iterasjoner [Fayad et al. 1999]. Fra problemkonkretisering, områdeanalyse, arkitekturdesign, implementasjon, uttesting, dokumentasjon, og forfra igjen (ikke nødvendigvis i samme rekkefølge) [Bosch et al. 1999]. Ny og ønsket funksjonalitet vil avsløres, og arkitekturen må endres og tilpasses [Roberts et al. 1996]. Denne oppgaven har i så måte bare besvart de tre første stegene i første iterasjon i Condors utvikling.

Bibliografi

[Abowd et al. 1996] G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, M. Pinkerton.
”CyberGuide: A Mobile Context-aware Tour Guide”.
Georgia Institute of Technology, Atalanta, 1996.

[Abowd 1998] G. D. Abowd.
”Software Design Issues for Ubiquitous Computing”.
Georgia Institute of Technology, Atalanta, 1998.

[Andersson et al. 1999] L. Andersson and Å. Rönnbom.
”Intelligent Agents - A New Technology for Future Distributed Sensor Systems?”
Department of Informatics, School of Economies and Commercial Law, Göteborg University 1999.

[Bosch et al. 1999] J. Bosch, P. Molin, M. Mattsson, P. Bengtsson, M. Fayad.
”Framework Problems and Experiences”.
Publisert i “Building Application Frameworks”, 1999, ISBN: 0-471-24875-4.

[Bjartveit et al. 1996] S. Bjartveit, T. Kjærstad.
”Kaos og kosmos”.
Oslo: Kolle forlag, 1996.

[Boone 1999] J. Boone.
”Harvesting Design”.
Publisert i “Building Application Frameworks”, 1999, ISBN: 0-471-24875-4.

[Brown 1995] P. J. Brown.
”The stick-e document: a framework for creating context-aware applications”.
The University of Canterbury, Kent CT2 7NF, UK., 1995.

[Brown 1998a] P. J. Brown.
”Some lessons for location-aware applications”.
The University of Canterbury, Kent CT2 7NF, UK., 1998.

[Brown 1998b] P. J. Brown.
”Triggering information by context”.
The University of Canterbury, Kent CT2 7NF, UK., 1998.

[Chen et al. 2000] G. Chen, D. Kotz.
”A Survey of Context-Aware Mobile Computing Research”.
Department of Computer Science, Dartmouth College, 2000.

[Cheverst et al. 2000] K. Cheverst, N. Davies, K. Mitchell, A. Friday, C. Efstratiou.
”Developing a Context-aware Electronic Tourist Guide: Some Issues and Experiences”.
Department of Computing, Lancaster University, 2000.

[Dey et al. 1998] A. K. Dey, G. D. Abowd, A. Wood.
”CyberDesk: a framework for providing self-integrating context-aware services”.
Georgia Institute of Technology, Atalanta.
School of Computer science. The University of Birmingham, 1998.

[Dey et al. 1999a] A. Dey, D. Salber. G. Abowd, M. Futakawa.
”The Conference Assistant: Combining Context-Awareness with Wearable Computing”.
Georgia Institute of Technology, Atalanta.
School of Computer science. The University of Birmingham, 1999.

[Dey et al. 1999b] A. Dey, G. Abowd.
”The Context Toolkit: Aiding the Development of Context-Aware Applications”.
Georgia Institute of Technology, Atalanta.
School of Computer science. The University of Birmingham, 1999.

[Dey 2000] A. K. Dey.
”Providing Architectural Support for Building Context-Aware Applications”.
Georgia Institute of Technology, 2000.

[Dix et al. 2000] A. Dix, T. Rodden, N. Davies, J. Trevor, A. Friday, K. Palfreyman.
”Exploiting Space and Location as a Design Framework for Interactive Mobile Systems”.
Lancaster University. ACM Transactions on Computer-Human Interaction, vol 7, No. 3, Sept. 2000, Pages 285-32.

[Davies et al. 2001] N. Davies, K. Cheverest, K. Mitchell, A. Efrat.
”Using and Determining Location in a Context-Sensitive Tour Guide”.
Publisert i Computer, vol. 32, nr. 8, August 2001.

[Efstratiou et al. 2001] C. Efstratiou, K. Cheverst, N. Davies, A. Friday.
”An Architecture for the Effective Support of Adaptive Context-Aware Applications”.
Multimedia Research Group, Department of Computing, Lancaster university, 2001.

[Fayad et al. 1999] M. E. Fayad, D. C. Schmidt, R. E. Johnson.
”Application Frameworks”.
Publisert i “Building Application Frameworks”, 1999, ISBN: 0-471-24875-4.

[Fontoura et al. 2002] M. Fontoura, W. Pree, B. Rumpe.
”The UML Profile for Framework Architechtures”.
ISBN: 0-201-67518-8

[Hong 2000] J. I. Hong.
”Context Fabric: Infrastructure Support for Context-Aware Systems”.
The University of California at Berkley, 2000.

[Jacobson et al. 1999] E. E. Jacobson, P. Nowack.
”Frameworks and Patterns: Architectural Abstractions”.
Publisert i “Building Application Frameworks”, 1999, ISBN: 0-471-24875-4.

[Kindberg et al. 2001] T. Kindberg, J. Barton, J. Morgan, G. Becker, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, H. Morris, J. Schettino, B. Serra.
”people, places, things: web presence for the real world”.
Hewlett-Packard Labratories. URL: Http://www.cooltown.com, 2001

[Klemke et al, 2001] R. Klemke, A. Nick.
”Case Studies in Developing Contextualising Information Systema”.
German National Research Center for Information Technology.

[Lieberman et al. 2000] H. Lieberman, T. Selker.
”Out of Context: Computer Systems That Adapt To, and Learn From, Context”.
Massachusetts Institute of Technology, 2000.

[Long et al. 1996] S. Long, R. Kooper, G. D. Abowd, C. G. Atkeseon.
”Rapid Prototyping og Mobile Context-Aware Applications: The Cyberguide Case Study”.
Georgia Institute of Technology, Atalanta, 1996.

[Miller et al. 1999] G. G. Miller, J. McGregor, M. L. Major.
”Capturing Framework Requirements”.
Publisert i “Building Application Frameworks”, 1999, ISBN: 0-471-24875-4.

[OALD 1990] Oxford Anvanced Learners Dictionary.
Oxford University Press 1990. ISBN: 0-19-431136-8.

[Pascoe 1998] J. Pascoe.
“Adding Generic Contextual Capabilities to Wearable Computers”.
Computer Laboratory, University of Kent at Canterbury, UK, 1998.

[Pree 1999] W. Pree.
”Hot-Spot-Driven Framework Development”.
University of Constance, Germany.
Publisert i “Building Application Frameworks”, 1999, ISBN: 0-471-24875-4.

[Roberts et al. 1996] D. Roberts, R. Johnson.
”Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks”.
University of Illinois 1996.

[Salber et al. 1998] D. Salber, G. D. Abowd.
”The Design and Use of a Generic Context Server”.
Georgia Institute of Technology, Atalanta.

[Schilit et al. 1994] B. N. Schilit, N. Adams, R. Want.
”Context-Aware Computing Applications”.
In Proceedings of the Workshop on Mobile Computing Systems and Applications, Santa *Cruz, CA, December 1994. IEEE Computer Society*.

[Schmid 1999] H. A. Schmid.
“Framework Design by Systematic Generalization“.
Publisert i “Building Application Frameworks”, 1999, ISBN: 0-471-24875-4.

[Sommerville 1998] I. Sommerville.
”Software Engineering”.
Fifth ediditon, ISBN: 0-201-42765-6, 1998

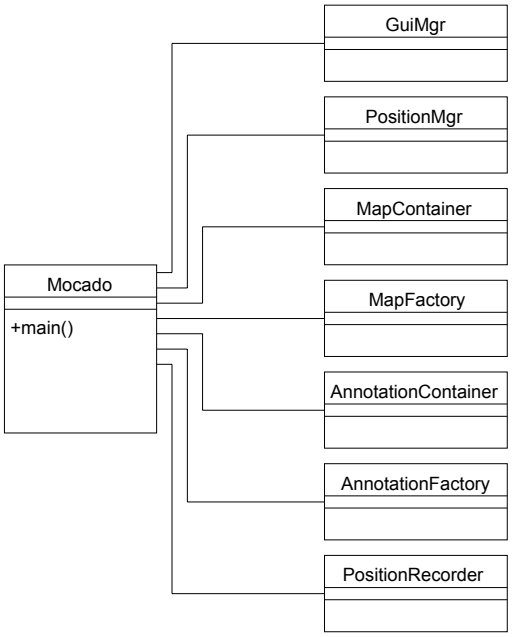
[Thanh 2001] Do van Thanh.
”Mobile Communications”.
University of Oslo - UNIK, 2001.

[Want et al. 2001] R. Want, B. Schilit.
”Expanding the Horizons of Location-Aware Computing”.
Publisert i Computer, vol. 32, nr. 8, August 2001.

Vedlegg

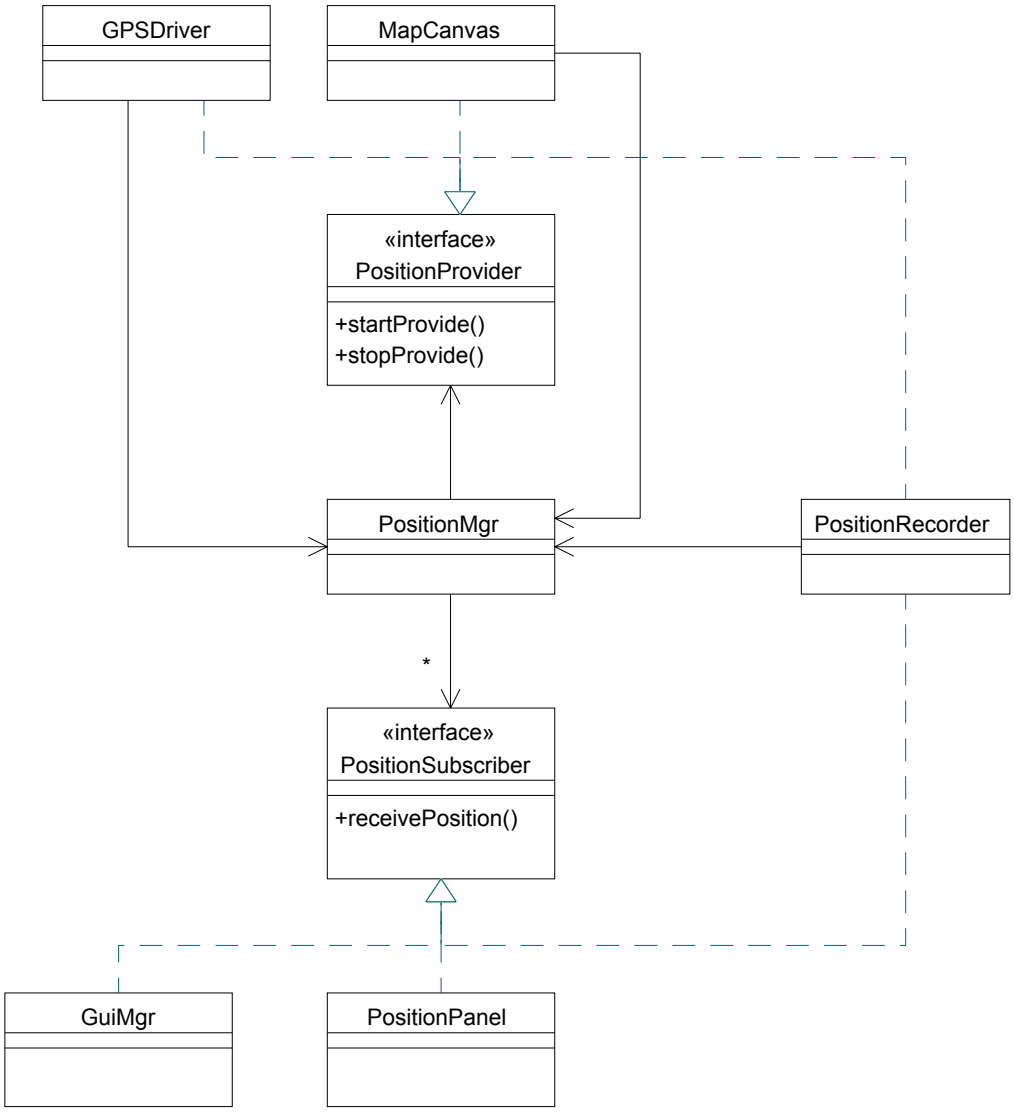
Jeg har valgt å legge ved UML-diagrammene for Mocado og Mobitras. Mye av hovedfagsoppgaven har bestått i å utvikle disse applikasjonene, og jeg vurderer av den grunn også deres arkitektur som interessante for leseren. Jeg vil ikke gå i detalj på hvordan arkitekturene jeg vil presentere virker, og de er heller ikke helt komplette. Alt som er relevant for oppgavens tema er imidlertid tatt med.

Mocado



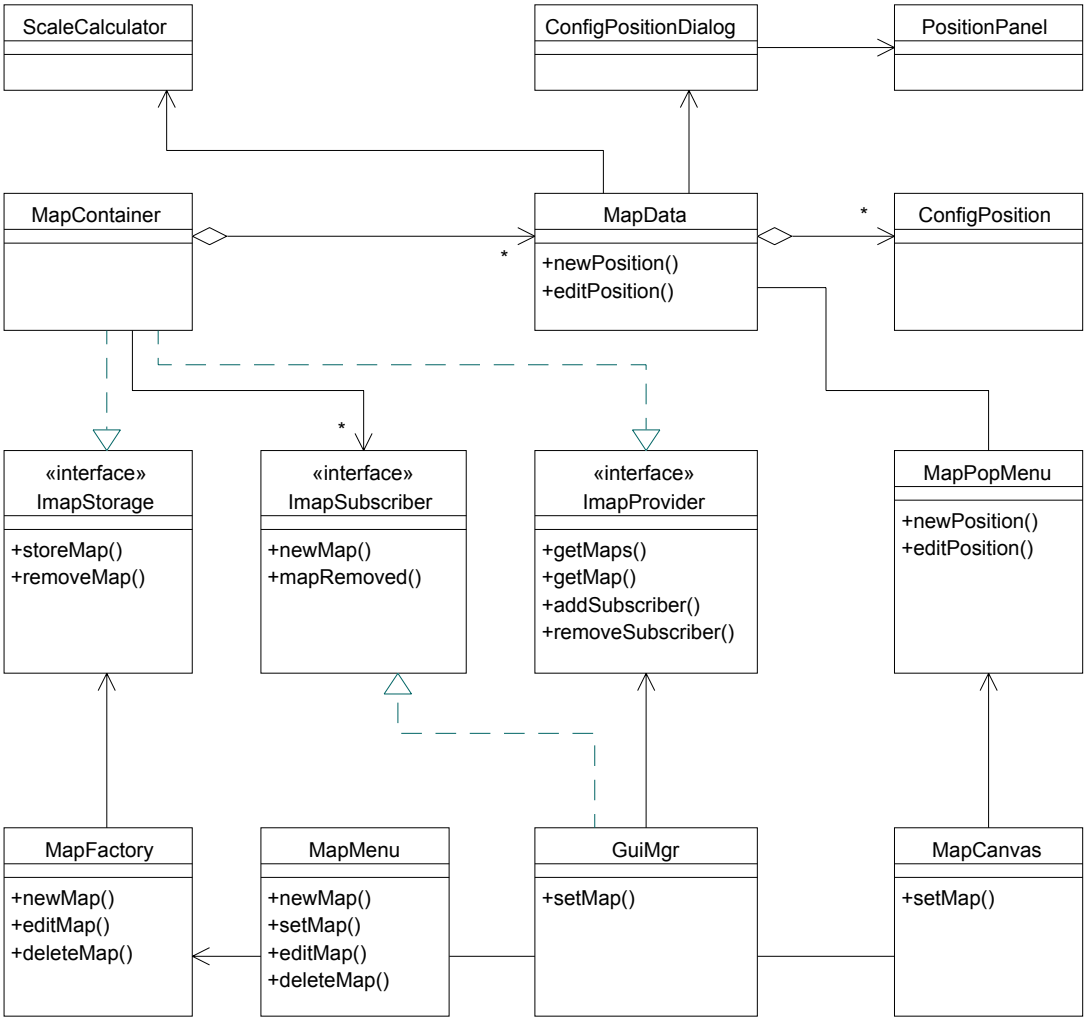
Figur 48: Klassen Mocado

Kommentar: Klassen *Mocado* er klassen som har ansvaret for initsialisere applikasjonen ved oppstart. Denne instansierer Mocados hovedklasser, og bygger de sammen på rett vis. Straks dette er gjort, overlates kontrollen til *GuiMgr*, som kommuniserer med brukeren ved hjelp av applikasjonens grafiske brukergrensesnitt.



Figur 49: Mocados posisjoneringsarkitektur

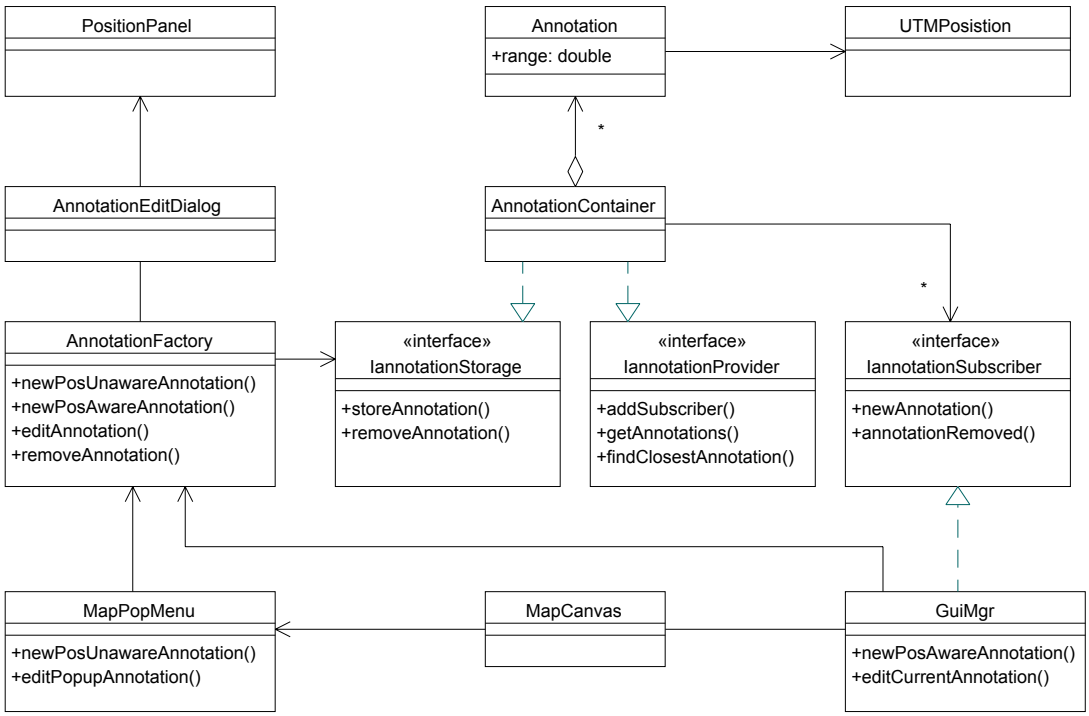
Kommentar: Arkitekturen viser hvordan Mocado registrerer posisjon. Sentralt i denne løsningen er *PositionMgr*. Denne fungerer som en mediator mellom alle som tilbyr, og alle som anvender posisjonsdata. Spesielt ser man at både *PositionRecorder*, *MapCanvas*, og *GPSDriver* tilbyr posisjon. Disse tilsvarer modiene ”Opptak-, og avspilling”, ”Manuell”, og ”GPS”, som beskrevet i Kapittel 5.5. Dette betyr at om brukeren har valgt GPS-modus, så kommer posisjonsdataene fra komponenten GPSDriver (som kommuniserer med GPS-mottakeren via PDA-ens kommunikasjonsport). Om han spiller av et opptak fra opptak-, og avspillingsfunksjonen, kommer dem isteden fra PositionRecorder. Om han trykker rundt på kartet i manuell modus, kommer dem fra GuiMgr. Arkitekturen viser også klassen PositionPanel. Denne kan motta opptil hundre posisjoner fra en av posisjonsleverandørene, og regne ut et gjennomsnitt av disse. Denne benyttes blant annet under kartkonfigurering.



Figur 50: Behandling av kart

Kommentar: Arkitekturen viser hvordan kart konstrueres og oppbevares i Mocado. Gangen for konstruksjon er slik: Først velges ”Create new map” av brukeren fra brukergrensesnittmenyen definert av klassen *MapMenu*. Dette resulterer i at funksjonen *newMap* i *MapMenu* blir kalt, og videre til at *newMap* i klassen *MapFactory* bli kalt. Denne tar seg av å konstruere en instanse av klassen *MapData*, og sende denne til metoden *storeMap* i *MapContainer*. MapData er klassen som representerer et kart, og inneholder blant annet dets konfigureringspunkter. Slike punkter blir representert ved hjelp av instanser fra klassen *ConfigPosition*.

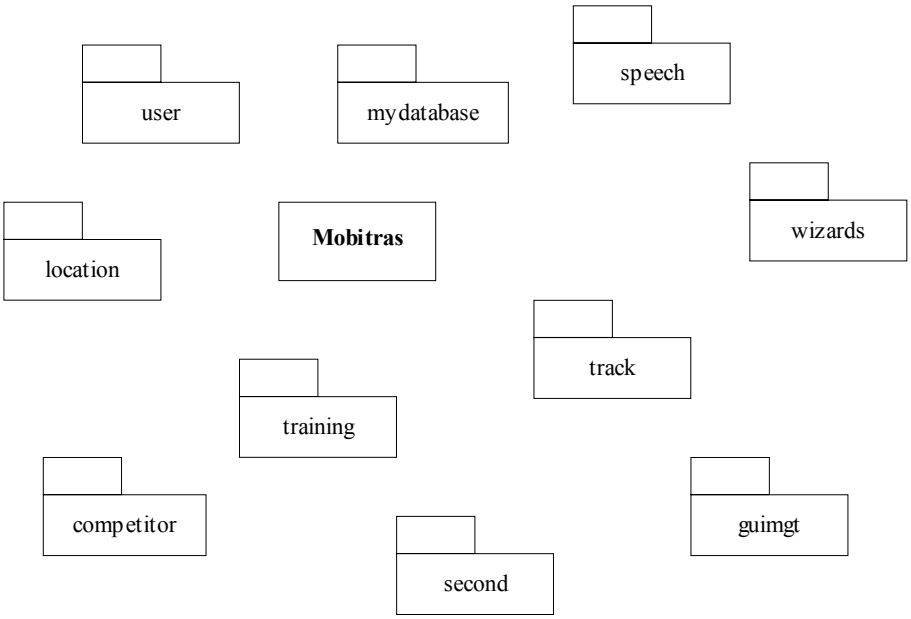
For å lage ett nytt konfigureringspunkt markers det på kartet hvor punktet skal ligge. Dette resulterer i at en pop-up dialog definert av klassen *MapPopMenu* vises (se Figur 8). Her kan brukeren velge å lage et nytt punkt, eller endre et som allerede eksisterende. MapCanvas har ansvaret for opptegning av kartet på skjermen sammen med brukerens posisjon og dets omkringliggende annoteringer.



Figur 51: Behandling av annoteringer

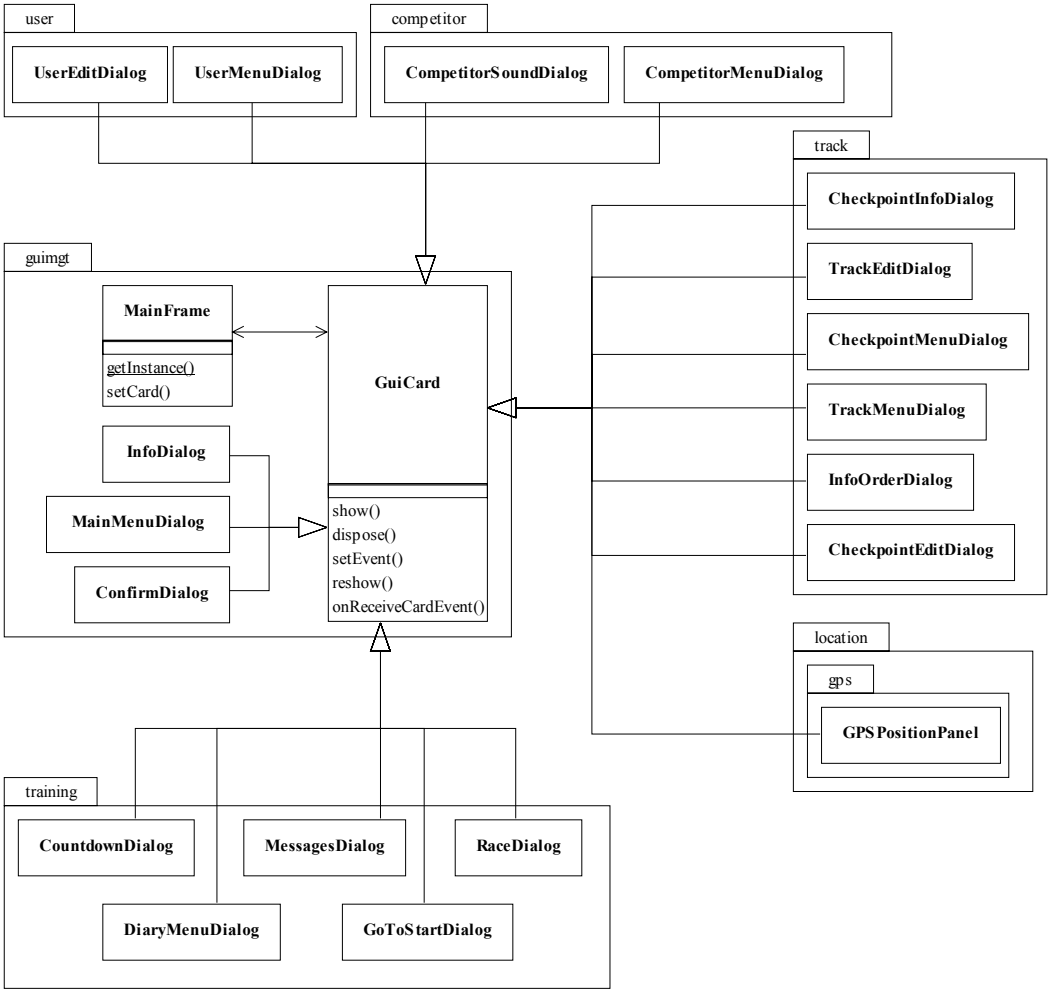
Kommentar: Arkitekturen viser hvordan annoteringer konstrueres og oppbevares i Mocado. En annotering blir representert av klassen *Annotation*, og inneholder en radius og en assosiasjon til en posisjon (*UTMPosition*). En annotering kan av brukeren konstrueres på to måter (jamfør 5.4): Enten ved å markere på kartet der den skal ligge, eller velge ”New” i menyen (se Figur 13). I det første tilfellet vises en dialogboks definert av klassen MapPopMenu, frem (se Figur 8). Om brukeren velger ”New annotation” her, blir først metoden *newPosUnawareAnnotation* i MapPopMenu kalt, deretter metoden med samme navn i klassen *AnnotationFactory*. Denne lager en instanse av klassen Annotation, og leverer den til *AnnotationContainer* for oppbevaring. Denne hendelsen blir også GuiMgr gjort oppmerksom på (via metoden *newAnnotation* i *IannotationSubscriber*), slik at annoteringen blir tegnet opp på kartet.

Mobitras



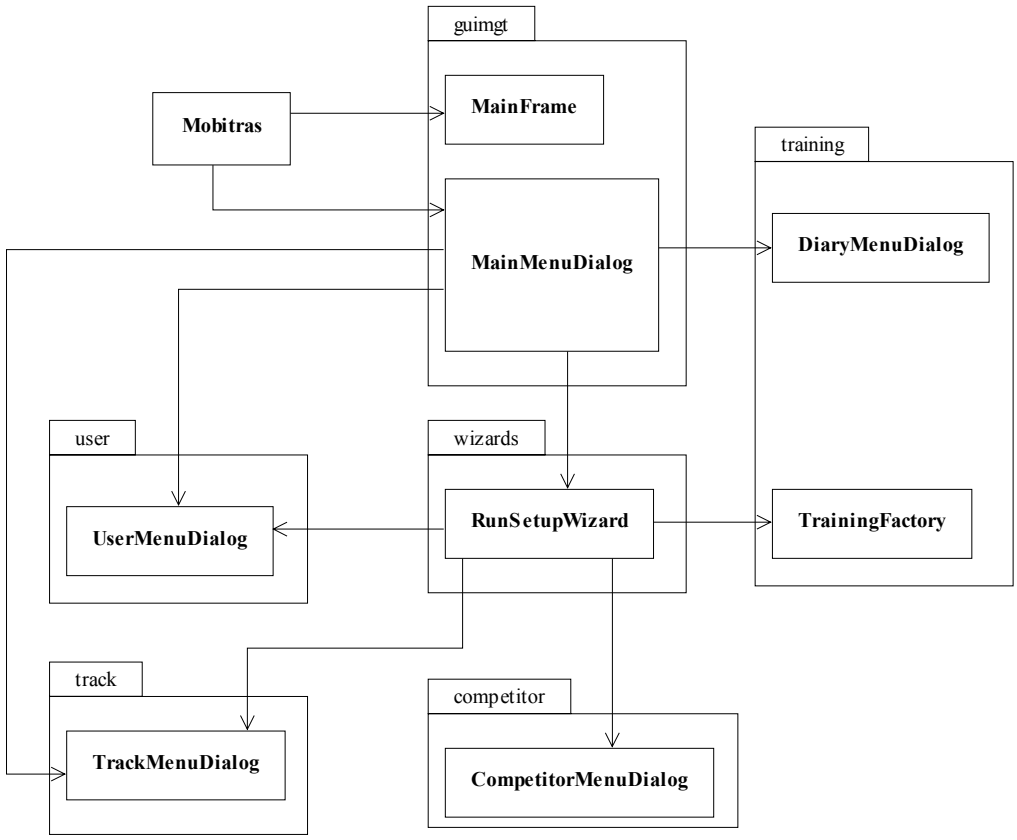
Figur 52: Mobitras komponentoversikt

Kommentar: Figuren viser en oversikt over alle komponentene som utgjør Mobitras. Klassen *Mobitras* (i figuren) er den som eksekveres utenfra, og som derfor inneholder metoden *main*.



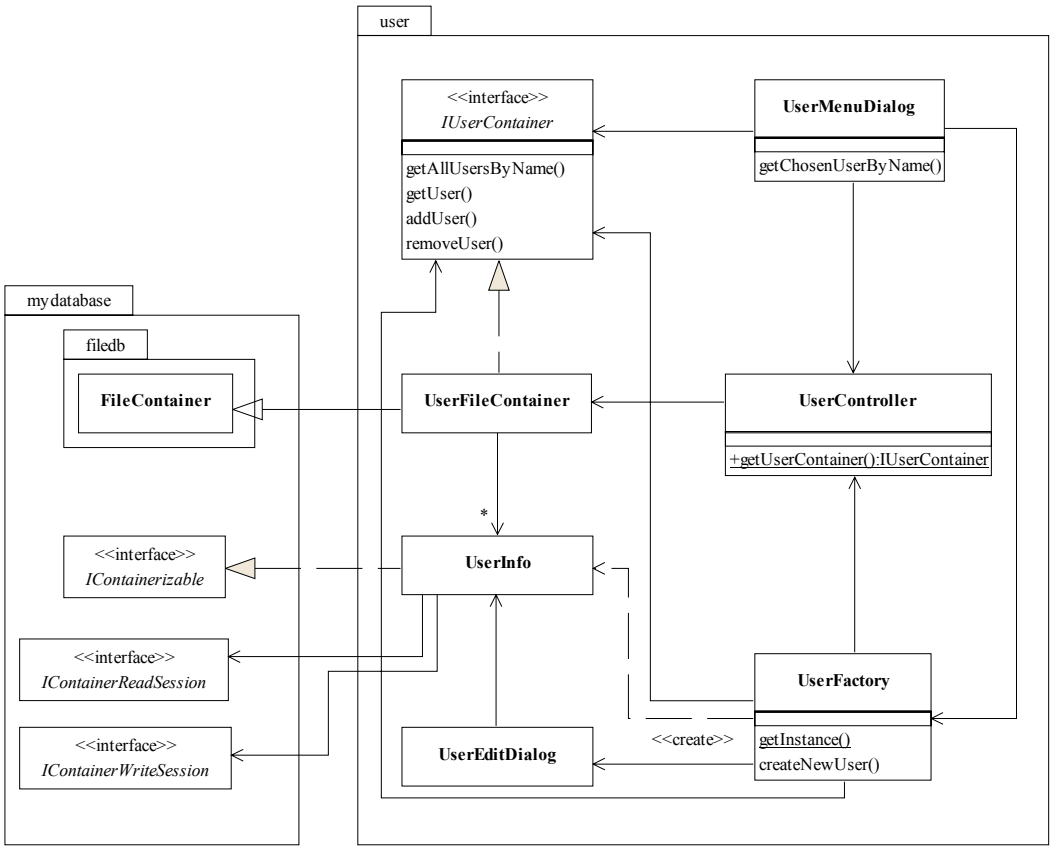
Figur 53: Mobitras brukergrensesnitt

Kommentar: Figuren viser en oversikt over hvordan det grafiske brukergrensesnittet i Mobitras er bygd opp. Alle dialogvinduer som brukeren kommuniserer med er realisert som spesialiseringer av *GuiCard*. GuiCard, igjen, inneholder alt av kildekode som er felles for hvordan dialogvinduene skal se ut og oppføre seg.



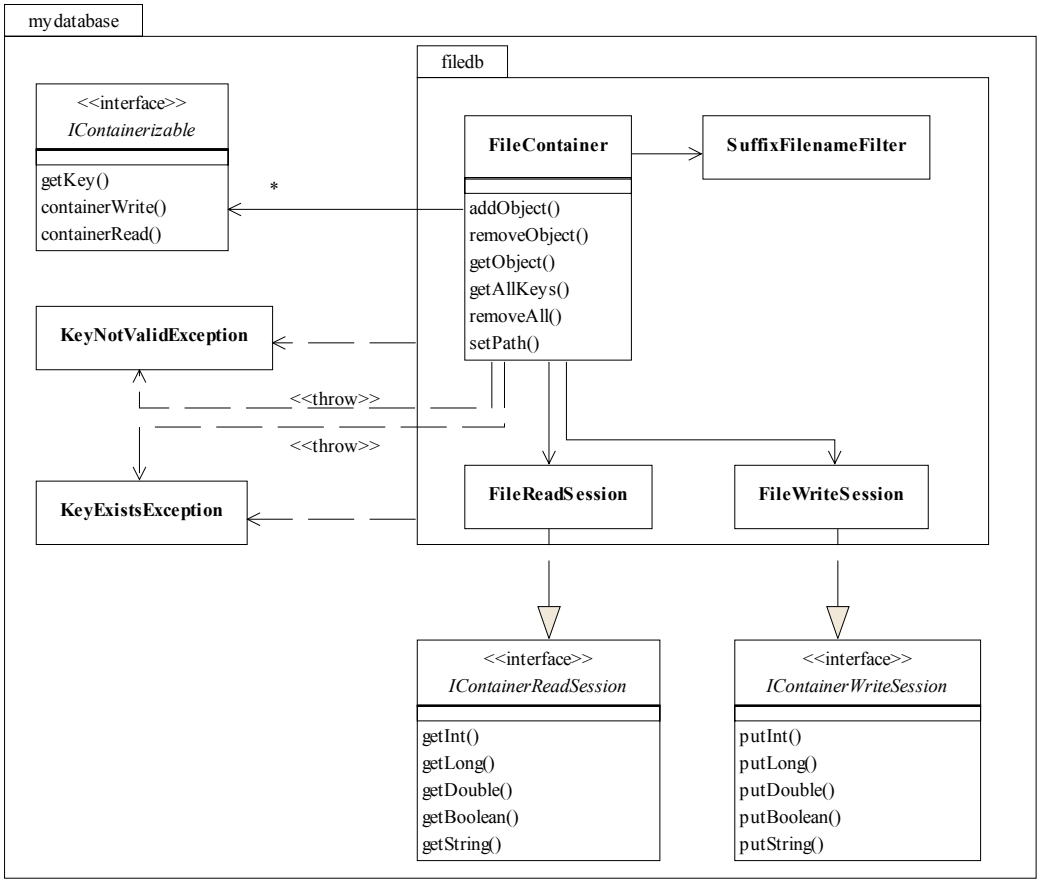
Figur 54: Mellomkomponent-kommunikasjon

Kommentar: Figuren viser assosiasjonene som i hovedsak benyttes når brukeren begynner en ny trening (se "Begin new training" i Figur 20). Denne dialogen tilsvarer *MainMenuDialog* på figuren, og er den første dialogen Mobitras oppretter ved oppstart. Herfra instansieres et objekt av typen *RunSetupWizard*, som veileder brukeren igjennom de resterende dialogene før løpet startes. Først velges brukerprofil i klassen *UserMenuDialog* (se Figur 19). Deretter velges løype i klassen *TrackMenuDialog* (se Figur 15). Videre velges skygger i dialogen *CompetitorMenuDialog* (se Figur 21). Til sist leveres alle de valgte dataene til klassen *TrainingFactory*, som overvåker selve treningen.



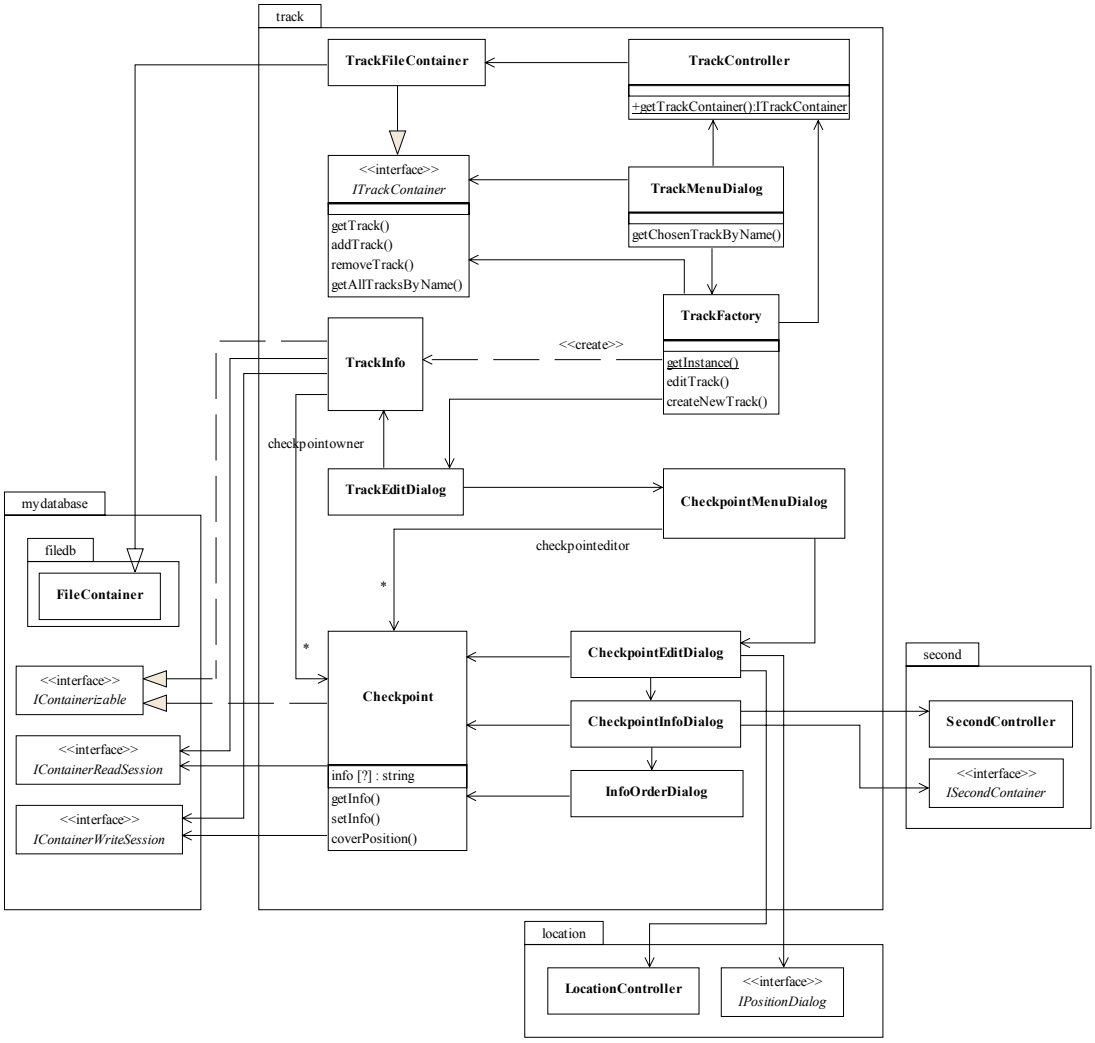
Figur 55: Behandling av brukerprofiler

Kommentar: Figuren viser to komponenter: *mydatabase*, og *user*. mydatabase er konstruert for å ivareta alt data en utøver produserer og konstruerer i applikasjonen. Denne benyttes flere steder i applikasjonen (se neste side). Den andre komponenten, user, viser klassene som anvendes for å behandle og ivareta brukerprofiler. *UserMenuDialog* er denne komponentens grensesnitt utad til resten av applikasjonen.



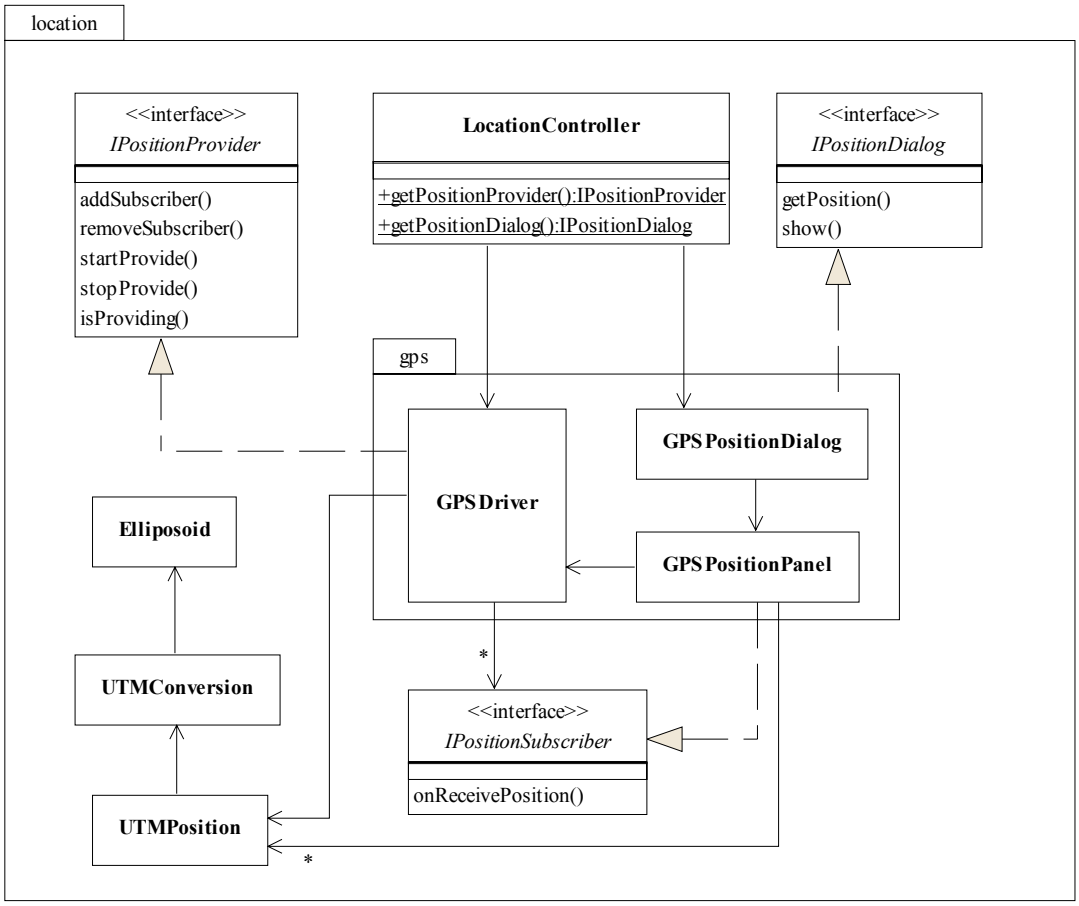
Figur 56: mydatabase

Kommentar: Figuren viser i mer detalj hvordan komponenten mydatabase ser ut på innsiden. Denne komponenten forsøker å skjule hvordan for eksempel bruker-profiler eller treninger oppbevares. Dette kan realiseres på flere ulike måter, og en konkret realisering som jeg har valgt, er komponenten *filedb*. Denne lagrer dataene ned på mange selvstendige tekstfiler. Dette systemet kan muligens virke noe unødig komplekst, men tanken har vært at man i senere versjoner kan ønske å unngå lagring av data lokalt på utøverens PDA, og heller velge å sende de over nettverket til en felles databaseserver. Grensesnittene som mydatabase tilbyr til resten av applikasjonen vil kunne tillate et slikt bytte relativt enkelt.



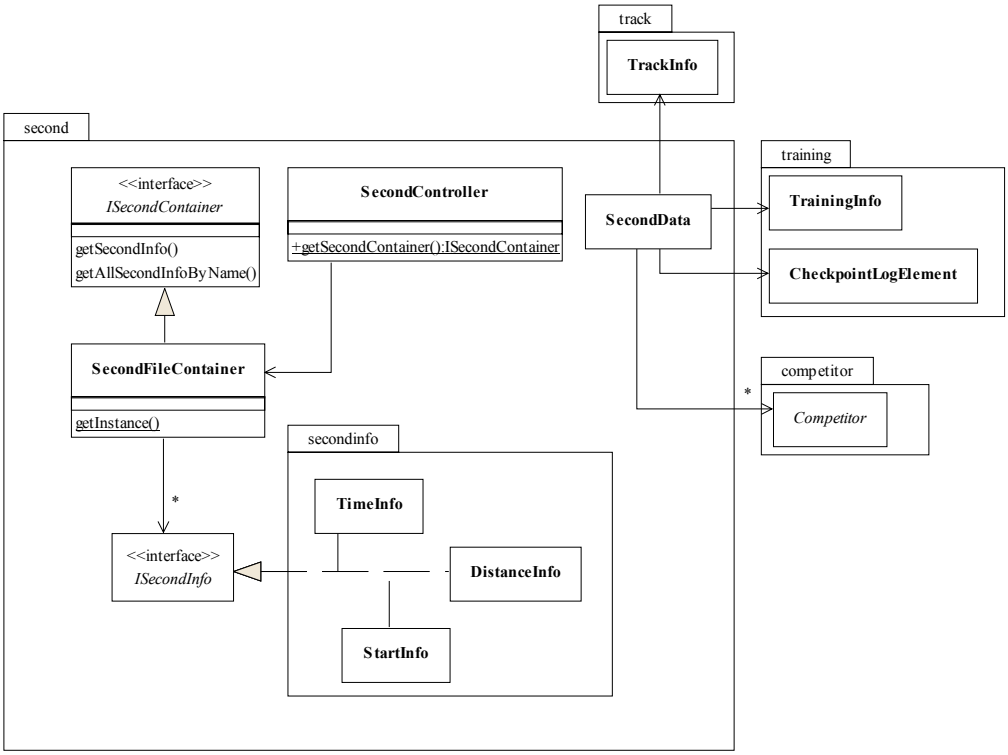
Figur 57: track-komponenten

Kommentar: Figuren viser hvordan komponenten *track* ser ut. Også denne benytter seg av mydatabase for å oppbevare konstruerte løyper. I tillegg anvendes komponenten *location* for å feste posisjonsangivelser fra GPS-mottakeren til nyopprettede sekunderingspunkter. Et sekunderingspunkt for en løype er representert ved hjelp av klassen *Checkpoint*, mens klassen *TrackMenuDialog* er komponentens grensesnitt utad (analogt med UserMenuDialog, se Figur 55). Komponenten *second* behandler løypens treningshendelser. Mer om disse komponentene vises i påfølgende diagrammer.



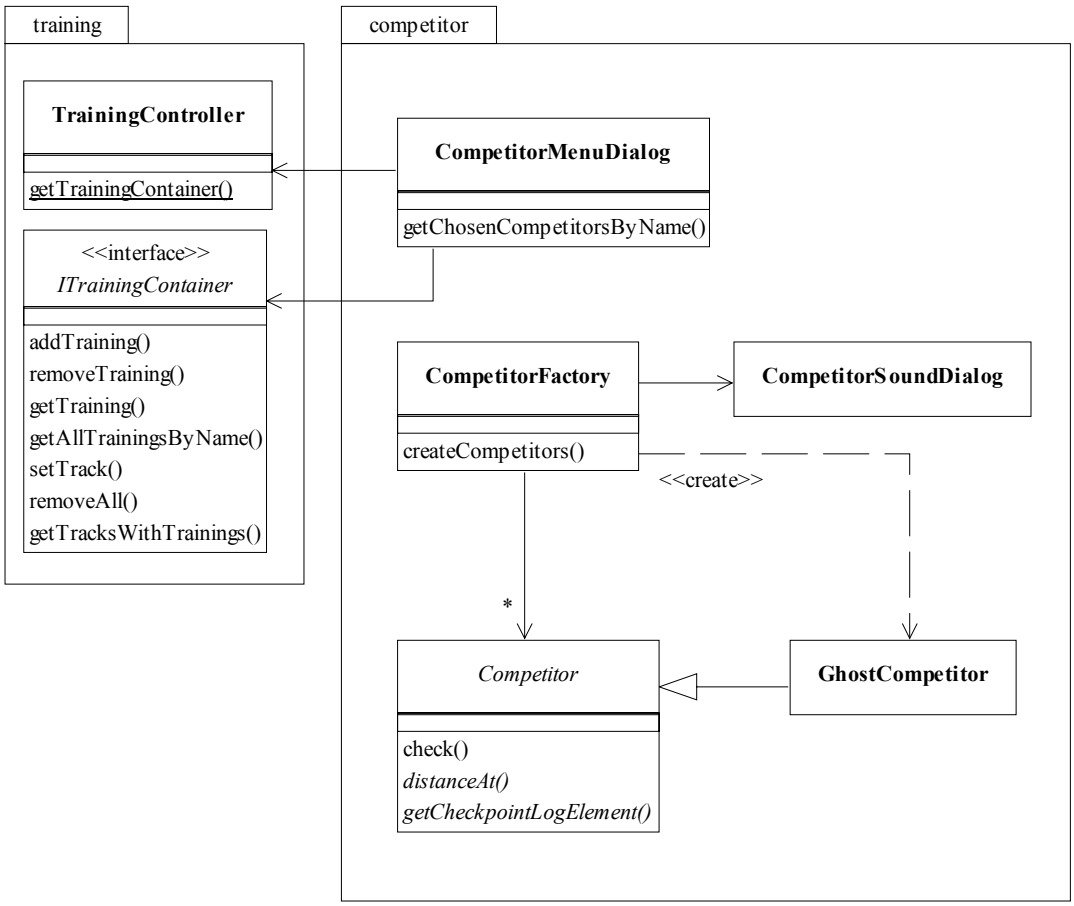
Figur 58: location-komponenten

Kommentar: Figuren viser hvordan komponenten location ser ut. Denne tar seg av å opprette kontakt med GPS-mottakeren og tilby omformede posisjoner til resten av applikasjonen. Denne komponenten forsøker også å skjule hvordan utøverens posisjon registreres. Selv om jeg i denne oppgaven har valgt å benytte GPS, trenger ikke dette nødvendigvis være tilfellet i fremtiden. Av den grunn er også komponenten ”gps” (som tar seg av å kommunisere med GPS-mottakeren), pakket inn bak posisjonsuniforme grensesnitt. Se også vedlegget sist i oppgaven for kildekoden til GPSDriver.



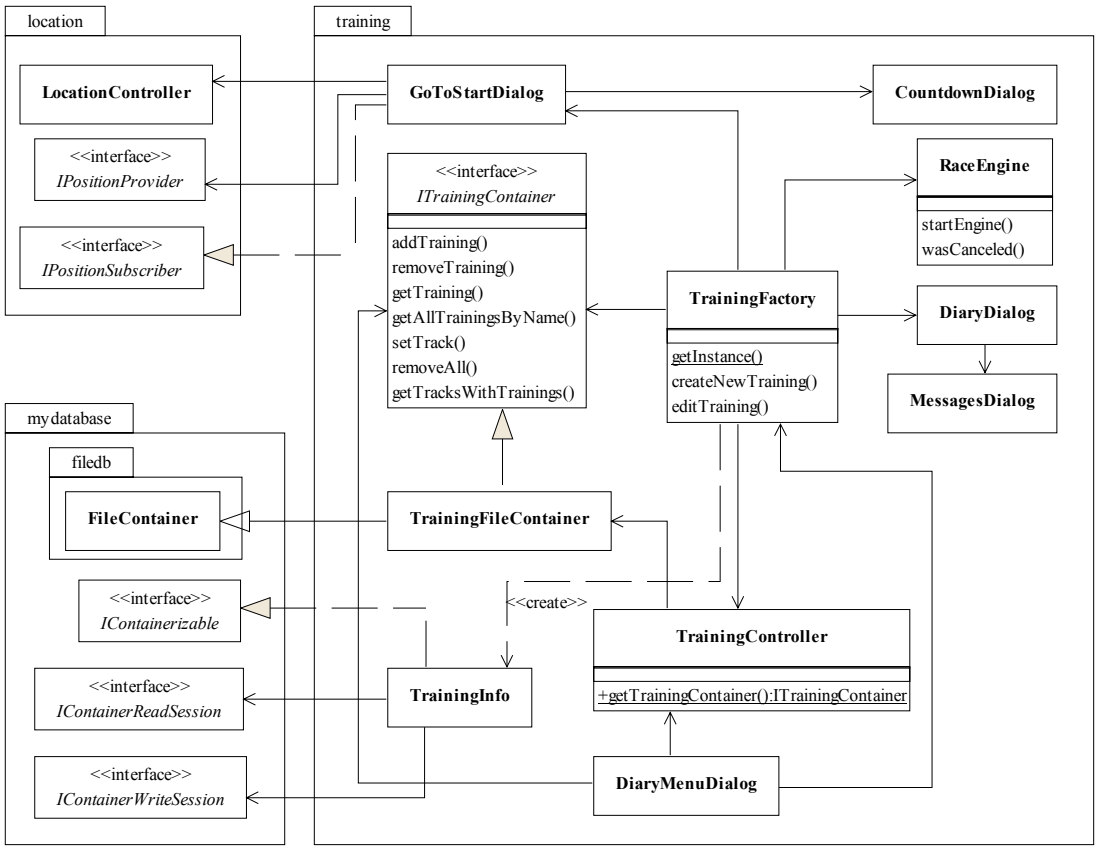
Figur 59: second-komponenten

Kommentar: Figuren viser komponenten second. Denne tar seg av behandlingen av kontekstrelevant informasjon. Slik informasjon benevnes her som *SecondInfo*, og inneholder blant annet klassen *StartInfo*. Denne klassen leser opp ordet ”start” når et nytt løp settes i gang. Klassen *SecondData* vil under en trening fylles med informasjon som beskriver utøverens kontekst. Dette inkluderer hans lokasjon, tid, distanse, skygger, passerte sekunderingspunkter og lignende. Dette leveres til løypens SecondInfo-klasser når informasjonen aktiveres. Dette tillater disse å lese opp informasjon som kan være basert på utøverens kontekstuelle situasjon (dynamisk informasjon).



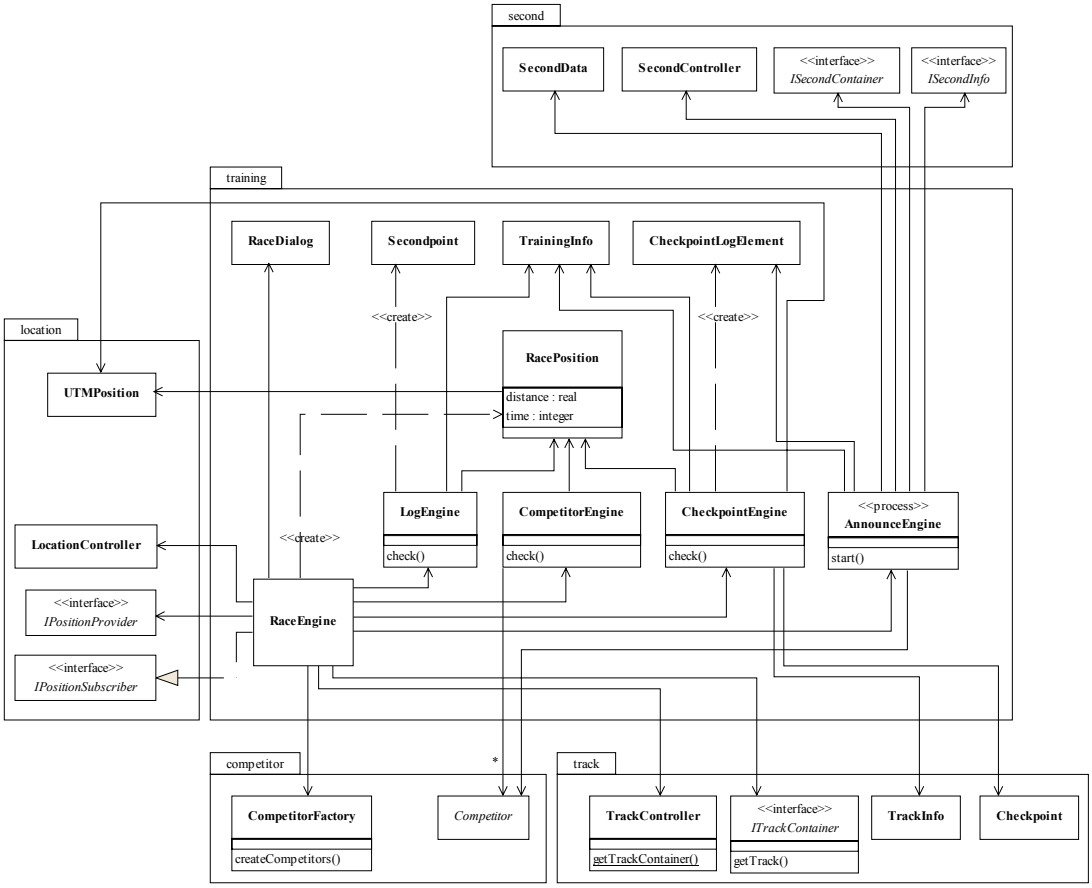
Figur 60: Skyggene

Kommentar: Figuren viser hvordan skygger behandles i Mobitras. Disse benevnes her som *competitors*. competitor-komponenten tar seg av å gjøre lagrede treninger om til skygger. Slike treninger blir hentet ut ifra komponenten *training*, som har ansvaret for å oppbevare og behandle disse.



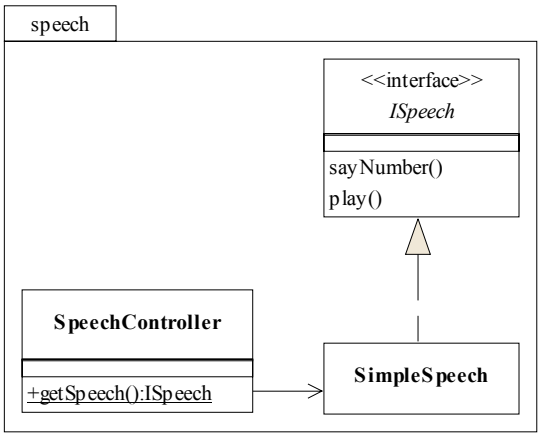
Figur 61: training-komponenten, del I

Kommentar: Figuren viser første del av hvordan komponenten training ser ut. Denne komponenten inneholder klassene som i hovedsak anvendes for å klargjøre en ny trening etter at utøveren har valgt brukerprofil, løype, og skygger. Denne klargjøringen er oppgaven til *TrainingFactory*, og startes ved å kalle på dens metode *CreateNewTraining*. Når brukeren har ankommet startområdet, og nedtellingen til start er ferdig (se GoToStartDialog), så overføres kontrollen (midlertidig) til *RaceEngine* i figuren på neste side.



Figur 62: training-komponenten, del II

Kommentar: Figuren viser hvordan komponenten training ser ut. Klassene som vises i dette diagrammet er de som i hovedsak benyttes under løpingen. Selve kontrollen er lagt til klassen RaceEngine. For hver nye posisjonsoppdatering denne mottar, vil den undersøke om et sekunderingspunkt er blitt passert, en treningshendelse har inntruffet, eller om utøveren har passert noen av skyggene. Dette er funksjonalitet som blir delegert til *CheckpointEngine*, og *Competitor-Engine*. Klassen *LogEngine* tar seg av å loggføre treningen (til treningsdag-boken), mens *AnnounceEngine* forteller utøveren med lyd om hendelsene som inntreffer underveis.



Figur 63: speech-komponenten

Kommentar: Figuren viser komponenten *speech*. Denne tar seg av å spille lyder for andre komponenter i applikasjonen. Spesielt er den også i stand til å lese opp tall mellom 0 og 10000. Blant annet SecondInfo-klassene benytter seg av denne komponenten flittig når sekunderingsinformasjon skal leses opp for utøveren.

Klassen GPSDriver

Denne klassen tar seg av å opprette kontakt med GPS-mottakeren og tilby brukerens posisjonering til resten av applikasjonen. Klassen tilsvarer GPSDriver hos Mobitras (jamfør Figur 58).

```
package location;
import javax.comm.*;
import java.util.*;
import java.io.*;
import java.util.Vector;

public class GPSDriver
implements SerialPortEventListener, IPositionProvider{

    private static GPSDriver instance;
    private SerialPort sPort;
    private InputStream commportstream;
    private boolean isproviding;

    private Vector subscribers;

    public static GPSDriver getInstance(){
        if (instance == null) instance = new GPSDriver();
        return instance;
    }

    private GPSDriver(){
        subscribers = new Vector();
    }

    // from IPositionProvider
    public void addSubscriber(IPositionSubscriber subscriber){
        subscribers.addElement(subscriber);
    }

    // from IPositionProvider
    public void removeSubscriber(IPositionSubscriber subscriber){
        subscribers.removeElement(subscriber);
    }

    // from IPositionProvider
    public void startProvide()
    throws Exception{
        openConnection();
    }

    // from IPositionProvider
    public void stopProvide()
    throws Exception{
        closeConnection();
    }
```

```
// from IPositionProvider
public boolean isProviding(){
    return isproviding;
}

public void distributePosition(UTMPosition value){
    for (int i=0; i<subscribers.size(); i++){
        IPositionSubscriber s;
        s = (IPositionSubscriber)subscribers.elementAt(i);
        s.onReceivePosition(value);
    }
}

public void serialEvent(SerialPortEvent e){
    if (e.getEventType() == SerialPortEvent.DATA_AVAILABLE){
        // Read data until -1 is returned.
        // If \r is received substitute
        // \n for correct newline handling.
        int newData = 0;
        StringBuffer inputBuffer = new StringBuffer();
        while (newData != -1)
        try{
            newData = commportstream.read();
            if (newData == -1) break;
            if ('\r' == (char)newData) inputBuffer.append('\n');
            else inputBuffer.append((char)newData);
        }
        catch (IOException ex){ return; }

        // search for locationdata...
        String input = inputBuffer.toString();
        String delim = String.valueOf('\n');
        StringTokenizer strTK = new StringTokenizer(input, delim);

        while (strTK.hasMoreTokens()){
            String temp = strTK.nextToken();
            if (temp.startsWith("$GPGLL")) handlePosition(temp);
        }
    }
}

private void handlePosition(String posString){
        try{
              StringTokenizer tok =
              new StringTokenizer(posString, ",");
              tok.nextToken(); //Throw away $GPGLL

              //Find the latitude information
              String latitude = tok.nextToken();
              int dotPos = latitude.indexOf(".");

              String latminString =
              latitude.substring(dotPos-2, latitude.length());
              double latMin =
              Double.valueOf(latminString).doubleValue();
              int latDeg =
              Integer.parseInt(latitude.substring(0, dotPos-2));

              tok.nextToken();
```

```
                //Find the longitude information
                String longitude = tok.nextToken();
                dotPos = longitude.indexOf(".");

                String longminString =
                longitude.substring(dotPos-2, longitude.length());
                double longMin =
                Double.valueOf(longminString).doubleValue();
                int longDeg =
                Integer.parseInt(longitude.substring(0, dotPos-2));


                UTMPosition retPos =
                new UTMPosition(longDeg, longMin, latDeg, latMin);
                distributePosition(retPos);
            }
            catch (Exception e){}
    }

    private void openConnection()
    throws Exception {
            if (!isproviding){
                try{
                    CommPortIdentifier portId =
                    CommPortIdentifier.getPortIdentifier("COM1");
                    sPort =
                    (SerialPort)portId.open("GPSDriver", 30000);
                    setConnectionParameters(sPort);
                    commportstream = sPort.getInputStream();
                    sPort.addEventListener(this);
                    sPort.notifyOnDataAvailable(true);
                    sPort.enableReceiveTimeout(30);
                    isproviding = true;
                }
                catch (Exception e){
                    if (sPort != null) sPort.close();
                    throw e;
                }
            }
    }

    public void setConnectionParameters(SerialPort sPort)
    throws Exception{
        sPort.setSerialPortParams(
            4800,
            SerialPort.DATABITS_8,
            SerialPort.STOPBITS_1,
            SerialPort.PARITY_NONE );
        sPort.setFlowControlMode(SerialPort.FLOWCONTROL_NONE);
    }

    public void closeConnection()
    throws Exception{
        if (sPort != null && isproviding) {
            commportstream.close();
            sPort.removeEventListener();
            sPort.close();
            isproviding = false;
        }
    }
}
```